В помощь радиолюбителю

Гололобов В. Н.

«УМНЫЙ ДОМ» СВОИМИ РУКАМИ

NT Press
Москва

УДК 621.38
ББК 32.965
Г61

Подписано в печать 22.08.2006. Фомат 84x108$^{1}/_{32}$. Гарнитура «Баскервиль». Печать офсетная. Усл. печ. л. 21,84. Тираж 3000 экз. Заказ № 5467.

Гололобов В. Н.

Г61 «Умный дом» своими руками / В. Н. Гололобов. – М. : НТ Пресс, 2007. – 416 с. : ил. – (В помощь радиолюбителю).

ISBN 5-477-00484-3

Книга адресована радиолюбителям, но может быть интересна всем, кто интересуется электроникой. Описывается создание системы автоматизации дома «Умный дом» на базе микроконтроллера PIC16F628A в программе MPLAB. Компоненты системы и модули отлаживаются на одной макетной плате. Для всех экспериментов, описанных в книге, можно использовать одну и ту же микросхему контроллера. Программатор, работающий с программой PonnyProg2000, легко собирается и не содержит дефицитных деталей. Компьютер в лаборатории радиолюбителя превращается, фактически, в саму лабораторию. Управляющая программа системы (она же – отладочная программа) может быть написана на языке программирования Visual Basic или на любом другом языке. В заключительной части приведены справочные материалы по командам микроконтроллера PIC16F628A, схема датчика движения и программатора для программирования PIC-контроллеров.

УДК 621.38
ББК 32.965



ОГЛАВЛЕНИЕ

Предисловие 6
Зачем нужна эта книга 6
Из чего мы будем создавать систему 8
Почему именно микроконтроллер в качестве базы 9
Еще немного о микроконтроллере в качестве базового элемента ... 10
Как мы будем работать 11
Несколько предварительных замечаний 11

Глава ■ 1
Базовая версия 12
«Умный дом» от AMX и JDS 12
Система Landmark 13
Система StarGate – X10 27
«Умный дом» вашего производства 33
Темный холл 33
Возвращаюсь я с работы 34
Возвращаюсь я с работы (модификация решения) 35
Создание эффекта присутствия 37
Цель проекта 40
Схема и программа релейного модуля 48
Программа модуля на ассемблере 51
Программа релейного модуля на языке С 66
Введение в работу с MPLAB 98
Релейный модуль, версия программы на языке С 104
Первая сборка на макетной плате 109
Схема и программа модуля приема ИК-команд 118
Программа модуля приема ИК-команд на языке С 122
Отладка модуля 154
Схема и программа модуля излучения ИК-кодов 166
Программа модуля излучения ИК-кодов на языке С 167
И что получилось? 201
Модуль считывания ИК-кодов WinLIRC 202
Программа для управляющего компьютера 208
Завтра 220
И немного назад 226
Текст основной программы на языке Visual Basic 228
Подведем итоги 234
Ода ошибкам 235

Глава ■ 2
Как расширить систему 237
Модуль цифровых вводов 239
Программа модуля цифровых вводов на языке С 241
Модуль с триаком 253
Модуль с плавной регулировкой яркости 256
Программа регулировки яркости на языке С 261
Модуль последовательного интерфейса 269
Модуль аудиокоммутатора 269
Модуль видеокоммутатора 270
Модуль управляемого усилителя 271
Модуль системного ИК-пульта управления 273
Модуль аналогового ввода для термометра 274
Замена проводного канала RS485 275
Усовершенствование базовых модулей 276
Последние замечания 277

Глава ■ 3
То, что рядом с «Умным домом» 280
MULTISIM 284
Усилительный каскад на транзисторе 289
CircuitMaker 2000 297
Electric 301
Сопряжение управления 306
·мешанные системы 315
азные подходы к реализации системы 318

Гриложение 330
ИК-датчик движения 330
Таблица команд микроконтроллера PIC16F628A 333
Цоколевка контроллера PIC16F628A 336
Программатор (совместно с PonyProg) 337
Адаптер для PIC-контроллеров 338
Внешний вид и параметры модуля
общего назначения фирмы Advantech 339
Практическое применение триака в модулях системы 339
Дополнительные замечания по ИК-управлению 341
Программа для компьютера в KDevelop 343
Вторая версия основной программы на языке С++ 355
Две полезные схемы 363
Разветвитель видеосигнала 363
Схемы для экспериментов с радиоканалом 367

Немного о программировании на C++ 372
Как писать программы на C++ 373
Определение и инициализация объектов данных 380
Написание выражений 385
Оператор предшествования 389
Написание условий и создание циклов 391
Циклы 396
Как использовать массивы и векторы 399
Указатели дают больше гибкости 404
Запись и чтение файлов 409
Ссылки на полезные сайты в Интернете 413

Предисловие

Зачем нужна эта книга

Многие не догадываются, насколько увлекательно занятие электроникой. Гораздо интереснее что-то придумывать и создавать, чем пользоваться готовым. Проделать предлагаемые в книге эксперименты и начать свою разработку? Это интересно. И может быть полезно.

Например. Заработавшись за компьютером дотемна, я часто, особенно зимой, выхожу по темной комнате в темный коридор, натыкаясь на телефонный столик и книжные шкафы. Используя релейный модуль, через который включена лампа на телефонном столике, я с компьютера включаю эту лампу, и теперь синяки от столкновения с мебелью полностью пропали.

Например. Я люблю, если чувствую, что устаю от работы, выпить чашечку кофе. Наливаю электрический чайник, включаю его. Чтобы не ждать, когда он закипит, я возвращаюсь к компьютеру. А когда вспоминаю о чайнике, его, как правило, приходится греть заново. Но система напомнит мне (если я не выключу звук у компьютера), что чайник вскипел.

Например. Возвращаюсь я с работы. Система встречает меня – зажигает свет в прихожей, кипятит воду для кофе (увы, чайник я наполняю утром). А когда я перейду в гостиную с чашкой кофе, она включит телевизор на программе новостей, чтобы я, усевшись в любимое кресло, посмотрел, что произошло в мире за день. Свет на кухне и в прихожей, который я, конечно, забыл выключить, она выключит сама.

Например. После вечерней работы я сажусь к телевизору в любимое кресло. Частенько бывает, что и вздремну. Очнувшись от дремы, чтобы не переломать сон, я быстро чищу зубы и спешу в кровать. Но не успеваю еще заснуть, когда система выключает телевизор и свет в квартире. Летом она

включает кондиционер, чтобы воздух в спальне стал прохладным. Но к утру, если температура ниже 24°, система включает обогрев. Если спать я люблю в прохладе, то вылезать из постели предпочитаю в теплую комнату.

Например. Мне нравится, уж так я устроен, принимать гостей. Люблю я суету в прихожей, когда, не успев поприветствовать одних своих знакомых и предложить им напитки, я спешу к двери на очередной звонок, чтобы встретить новых путников, забредших «на огонек». И не слишком, как ни стараюсь, я поспеваю всех развлечь. Если бы не система, которая и свет во время зажигает, и музыку включает негромко, и тем, кто зашел в мой кабинет, предлагает посмотреть фильм.

Все эти и многие другие «**Например**» могут быть реализованы из доступных деталей своими руками с помощью системы, описанной в книге. И ничто не помешает применить эти разработки не только в квартире, но и на даче, в гараже или в машине.

Мне кажется, это интересно – придумывать, что бы еще система могла сделать полезного? Экспериментировать, а затем воплощать проекты в жизнь.

Строить всегда интересно. Строишь ли ты дом или планы, отношения с коллегами по работе или свою первую электронную схему. Как мне кажется, электроника – самая удобная сфера применения творческих способностей. Особенно современная электроника.

Своими успехами электроника во многом обязана новым технологиям. В частности, появлению компьютерных программам, позволяющих осуществлять полную разработку устройств за компьютером. Конечно, чтобы осуществить такую разработку, нужно иметь компьютер. Но если для профессиональной работы следует обзаводиться одной из последних моделей, то радиолюбитель может прекрасно поладить с ранними моделями, стоимость которых может сравниться со стоимостью приборов в лаборатории радиолюбителя. Любительский осциллограф (я не говорю о профессиональном) стоит столько же, а то и дороже, чем подержанный Pentium с тактовой частотой 120 МГц. Сами программы (некоторые из них достаточно дорогостоящие) могут быть в демо-версиях,

которые имеют основное ограничение – невозможность сохранения файла. Я сам порою пользуюсь подобными версиями, и это, согласен, неудобно, но если потратить чуть больше времени на продуманный план работы, подобное неудобство перестает быть определяющим. Кроме того, для компьютера сегодня есть операционная система Linux. Одним из основных ее достоинств является доступность по цене и существование большого количества программ, распространяемых бесплатно.

Систем «Умный дом», то есть систем автоматизации жилья, в настоящее время множество. Одни рассчитаны на применение в условиях квартиры, другие могут поддерживать работу целых жилых районов. Хотя интересы радиолюбителя могут простираться далеко за пределы его мастерской (у меня – это секретер, у кого-то, может быть, целая комната), начинать эксперименты лучше на столе.

Итак, прежде чем начинать работу, определимся в том, **что мы намерены сделать**. Небольшую систему, к которой, пусть с оговорками, можно применить название «Умный дом». Начав с разработки отдельных компонентов системы, мы объединим их в сеть, заставив работать под управлением компьютера. В первом приближении так построены и многие коммерческие варианты системы, за исключением, может быть, специализированного процессора или микроконтроллера в качестве центрального управляющего устройства. Но кто мешает нам загрузить небольшую системную программу в контроллер? Или что может помешать построить модули так, что они будут «общаться» между собой, не требуя «ценных указаний» от компьютера?

Из чего мы будем создавать систему

Из того, что есть «под рукой». Что доступно.

В качестве базового элемента всей системы я предлагаю использовать микроконтроллер фирмы Microchip PIC16F628A. Причина, по которой я остановил свой выбор на этом микроконтроллере, весьма прозаична – стоимость и доступность как самого контроллера, так и всего, что необходимо, чтобы микросхема, которую вы купили, могла быть чем-то полезна. То есть программатор и программа обслуживания программатора,

которые позволят загрузить в микросхему вашу разработку. Программа для работы с микросхемой PIC16F628A – это MPLAB. Она свободно распространяется изготовителями микросхем. Программатор используется самодельный, работающий с программой PonyProg 2000, которая также распространяется свободно. Все это программное обеспечение существует как для Windows, так и для Linux.

Почему именно микроконтроллер в качестве базы

Не только и не столько «в погоне за модой», но по нескольким иным причинам. Это удобно и практично. Создав одну из конструкций, описанных в книге, вы легко превращаете ее в другую, не докупая новых дорогостоящих элементов. Потеряв со временем интерес к данной теме, вы можете не отправлять макетную плату с микроконтроллером в ящик с надписью «Хлам», а использовать ее для построения ряда полезных схем в следующем проекте или сделать микроконтроллер ядром следующей системы. Современные микроконтроллеры – это целый мир электроники, рождающейся из одного элемента. Очень интересной темой, как мне кажется, могла бы стать тема модификации старых конструкций с использованием микроконтроллера. Микроконтроллер PIC16F628A имеет встроенные блоки компараторов и широтно-импульсной модуляции, а также энергонезависимый модуль памяти. Его тактовая частота может быть повышена до 20 МГц, а выполнение ряда команд за один такт позволит применять микросхему в диапазоне радиочастот.

Есть и еще одна причина, заставившая меня написать эту книгу, – обилие новых терминов может оттолкнуть начинающего. Человек, незнакомый с программированием, может загодя, не пытаясь разобраться, решить, что программирование – не для него. Я же пытаюсь показать (или доказать), что все не так. И программы, которые вы будете использовать, и языки программирования – это те же молоток или отвертка, только называются они иначе. Это инструменты. Ведь не отталкивают вас такие термины, как мультиметр, осциллограф, функциональный генератор, варикап, термистор. Это все средства достижения вашей цели.

В настоящий момент микросхему PIC16F628A в Москве можно купить в магазине «Чип и Дип» или заказать по почте наложенным платежом на сайте http://www.dessy.ru. Забегая немного вперед, хочу отметить, что я не пожалел о выборе микросхемы, – при налаживании устройств мне пришлось сотни раз перепрограммировать ее. Теряя бдительность, я несколько раз устанавливал ее в панельку неправильно, но она как работала, так и работает. Одно это, как мне кажется, достойно восхищения и уважения. В качестве базового элемента всех предлагаемых разработок можно использовать любой другой микроконтроллер, но это потребует изменения всего комплекса программ и программатора. Если по какой-либо причине, кроме стратегической, есть необходимость сменить микросхему, попробуйте выбрать ее из серии PIC, поскольку со многими микросхемами этой серии работают программаторы и программы PonyProg2000 и MPLAB.

Еще немного о микроконтроллере в качестве базового элемента

Когда-то основным элементом при построении схем была вакуумная лампа. С появлением полупроводников транзисторы почти вытеснили ламы. Микросхемы, укрывая в своих глубинах тысячи транзисторов, изменили подход к разработке электронных изделий. Все в большей мере электронные изделия стали превращаться в кентавра – наполовину транзисторы-резисторы, наполовину программы. Микроконтроллер устранил и эту половинчатость. Он – микросхема, работающая на основе написанной для нее программы, которая и определяет все, что микросхема будет делать (в рамках, конечно, своих физических возможностей). Сегодня микроконтроллеры используются настолько же широко, насколько вчера использовались транзисторы, а позавчера – лампы. Стоимость микросхемы PIC16F628A (в пластмассовом корпусе DIP18) – около 100 руб. С одной единственной микросхемой вы можете собрать, проверить, отладить и модифицировать все модули, описанные в книге. Вы можете придумать свои модули и проверить их работу, а также загрузить основную программу в микроконтроллер и, используя компьютер

в качестве других модулей системы, проверить работу основной программы. Для проверки всех схем можно использовать одну макетную плату, добавляя элементы по мере необходимости.

И последнее – если вы после макетирования и отладки собрали готовый вариант, спаяв схему полностью, но в процессе эксплуатации нашли изъяны в ее работе, можно перепрограммировать микроконтроллер в готовой схеме, не выпаивая его.

Как мы будем работать

Технология проста – в основном, за компьютером. Когда покажется, что разработка готова, мы перенесем ее на макетную плату. После отладки разработки на макетной плате ее можно, при желании, повторить в виде «готового изделия» с этикеткой «Made in MyLab». При налаживании всех схем я намеренно постарался использовать единственный невиртуальный прибор – мультиметр. Для этих целей подойдет самый дешевый цифровой мультиметр, и мне жаль, что я не проверил это, но думаю, подойдет и тестер класса «Приз» или ТЛ-4.

Несколько предварительных замечаний

«Умный дом», «Смышленый дом», или «Послушный дом» – за всеми этими названиями будет скрываться ваше желание применить все, о чем говорится в книге, используя собственные возможности и фантазию, к комнате, квартире, коттеджу или дому. «Умный дом» будет умен ровно настолько, насколько вы ему позволите.

Основная цель этой книги – не столько в описании готовых разработок, хотя все завершенные устройства проверены в макетном варианте, сколько в рассказе об одном из путей подхода к разработке. Кто-то из вас уже имеет опыт работы с микроконтроллерами. Но кто-то, возможно, не пытался строить свои разработки на базе микроконтроллера, полагая, что это слишком сложно. Однако работать с микроконтроллером не сложнее, чем с любыми электронными компонентами. Было бы время и желание.

Глава 1

Базовая версия

«Умный дом» от AMX и JDS

Сколько специалистов, столько мнений. Я часто повторяю это, поскольку решений может существовать множество, даже после применения всех критериев отбора. Дальнейший выбор происходит на основе личных предпочтений. Системы автоматизации быта – отнюдь не исключение. Можно до бесконечности спорить, делать ли систему централизованной или децентрализованной, какую сеть использовать – компьютерную, силовую или специализированную. За основу выбора можно взять надежность, стоимость или доступность готовых устройств, за счет которых в будущем система может расширяться, совершенствоваться, развиваться.

Полный обзор существующих систем может занять не одну книгу, поэтому я лишь вкратце расскажу о нескольких системах, с которыми знаком, и которые находятся, в какой-то мере, на ценовых полюсах систем бытового назначения. Но вначале немного о том, из чего состоит любая система «Умный дом», хотя можно по-разному подойти и к этому вопросу. Я разделю систему на центральное управляющее устройство, исполняющие модули, средства управления, системную сеть и среду программирования (она же средство отладки). Базовые исполняющие модули – релейные модули, модули цифровых вводов, диммеры, трансляторы ИК-команд, коммутаторы сигналов. Средства управления – универсальные пульты ИК (инфракрасный спектр) команд, специализированные

клавишные пульты и сенсорные панели. Системная сеть – это системный интерфейс и среда передачи системных команд. Это не полный перечень, а, скорее, произвольная выборка, которая нужна в дальнейшем, чтобы определиться с реализацией.

Кроме того, все производители систем каждодневно совершенствуют все составляющие своих продуктов. Не успеваешь рассказать о новинках, предлагаемых фирмой, как она выпускает нечто еще более интересное и привлекательное. Вместе с совершенствованием информационных технологий преобразуются подходы к решениям, да и производители бытовой техники все чаще задумываются о возможности включения своих изделий в единую систему, снабжая их стандартными интерфейсами.

Система Landmark

В настоящее время система поддерживается корпорацией AMX (PHAST, Panja).

Почти все модули выполнены в виде печатных плат, предназначенных для установки в конструктив (рис. 1.1).



Рис. 1.1. Конструктив PHAST для установки модулей

С перечнем модулей можно ознакомиться на сайте производителя, но я приведу названия и назначение некоторых модулей из списка, который есть в моем архиве:

- PLC-MCU – модуль центрального процессора системы;
- PLB-AMP8 – 8- канальный усилитель (обслуживание акустических зон);

- PLB-AS16 – 16-канальный аудио коммутатор (матрица 16×16);
- PLC-IN7 – модуль с семью цифровыми входами (для подключения датчиков);
- PLC-IRIN – модуль с тремя входами для приемников ИК управляющих кодов;
- PLC-IROUT – модуль для подключения 4-х ИК-излучателей;
- PLC-RL8 – релейный модуль с восьмью исполняющими реле;
- PLL-MLC – управляемый диммер-выключатель.

В конструктиве (CardFrame) расположен центральный процессор. Конструктив имеет встроенный сетевой концентратор, к которому подключаются сетевые системные устройства (например, все выключатели света). Как и компьютерные хабы, он имеет вход и выход для объединения всех сетевых концентраторов.

Средства управления в составе системы представлены двумя базовыми решениями. Это настенный клавишный пульт с дисплеем для отображения информации и переносной пульт для ИК-управления. Кроме них система поддерживает работу сенсорных панелей AMX с помощью специального модуля. Позже появилось множество универсальных ИК-пультов управления, способных запомнить большое количество кодов от пультов управления бытовыми устройствами. Некоторые их них, например фирмы Philips, могут полностью программироваться на компьютере, а затем загружаться через COM или USB-порт.

Несомненно, и сенсорные панели AMX, и пульты Philips очень красивы (рис. 1.2)

Вид и работа сенсорных панелей обустраиваются с помощью специальной программы (или программ), которые в полной мере определяют все необходимое, подготавливая рабочую программу для загрузки в панель. Панели также имеют встроенный редактор, но программировать их удобнее все-таки за компьютером. Для загрузки панелей, уже установленных в систему, используется существующая системная сеть. То же можно сказать и о пультах фирмы Philips (рис. 1.3).

Рис. 1.2. Сенсорная панель AMX



Рис. 1.3. Пульт ProntoPro

Универсальные пульты для ИК-управления могут применяться, практически, с любой системой. Некоторые модели поддерживают как инфракрасное управление, так и управление по радиоканалу. Например, пульты ProntoPro позволяют одни страницы использовать в режиме IR (инфракрасное

управление), а другие – в режиме RF (управление по радиоканалу). Какой из режимов удобнее в практическом применении, сказать трудно. С одной стороны, ИК-управление ограничено помещением, в котором оно используется, а управление по радиоканалу не имеет подобного ограничения. Но с другой стороны, именно возможность ограничить область управления оказывается очень важным свойством. В данном случае наилучшее решение зависит от конкретной задачи и условий применения. Да и не следует переоценивать проникающую способность радиоволн. Серьезным препятствием на их пути может стать развитая силовая сеть или система отопления с большим количеством металлических труб, скрытых в стенах.

Система Landmark имеет удобную и эффективную среду программирования, позволяющую работать как с проектом автоматизации обычной квартиры, так и с автоматизацией многоэтажной постройки. Каждый этаж в последнем случае может быть разбит на помещения, в которых будут располагаться устройства и датчики, бытовая аппаратура. Можно использовать строительные поэтажные планы, что дает полное представление о помещениях.

Программа, по сути, набирается из необходимых событий, вызываемых пультами управления, датчиками, или временем суток, и ответных действий системы. Программа оснащена достаточным количеством условий.

Приведу пример работы с системой Landmark. После запуска и ввода пароля новый проект выглядит, как представлено на рис. 1.4.

Первые клавиши инструментальной панели, ниже основного меню, вполне ясно подскажут, что делать дальше, – определиться с домом – этажами и комнатами. Рисунок этажа можно взять из проектной архитектурной документации, изобразить в редакторе Paint, встроенном в Windows, в AutoCAD или CorelDraw. Landmark поддерживает множество форматов изображений. Щелкнув кнопку в правом углу инструментальной панели, выбрав имя для этажа (поддерживается русский для задания названий, но лучше использовать английский, чтобы при печати иметь как можно меньше проблем) и указав рисунок этажа, получаем результат, показанный на рис. 1.5.

Рис. 1.4. Окно проекта в системе Landmark

Рис. 1.5. Этаж в системе Landmark

На этом рисунке мы расположим помещения проекта, перейдя на вторую закладку основной панели инструментов «Rooms». Для добавления помещения используем клавишу «Add» меню страницы. Впишем название комнаты. Можно изменить цвет, а можно оставить цвет, предлагаемый программой. Затем нарисуем помещения, используя левую кнопку мыши для фиксации положения линии. Если впоследствии понадобится поправить положение зафиксированной точки, можно установить курсор на эту точку и, удерживая левую кнопку мыши, переместить опорную точку в нужное место.

Продвигаясь дальше по основной инструментальной панели, можно понять, что необходимо сделать, поскольку следующая кнопка носит название **Design**.

А пока вид проекта в плане помещений представлен на рис. 1.6.

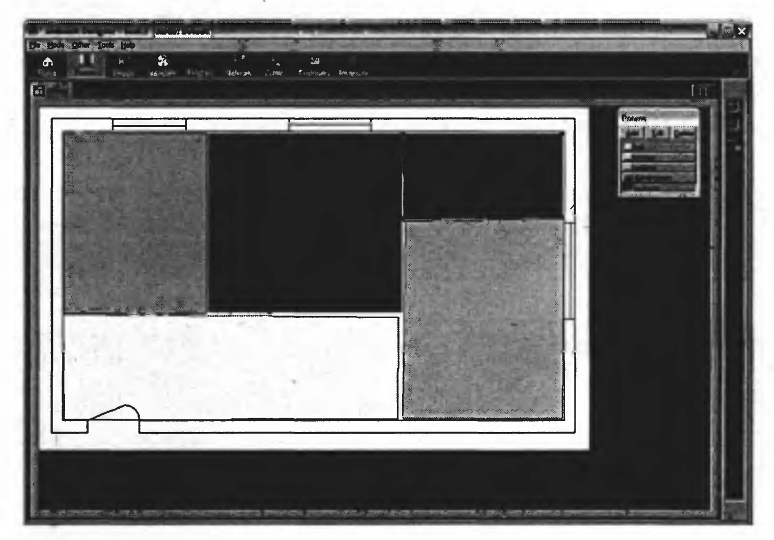


Рис. 1.6. Разбивка проекта на комнаты

Если теперь перейти на вкладку **Design**, получим вид проекта, изображенный на рис. 1.7.

На инструментальной панели страницы (справа) все устройства, поддерживаемые системой. Их нужно расположить по комнатам, в соответствии с реальным положением дел.

Рис. 1.7. Проект, подготовленный к размещению оборудования

Для начала расставим свет (клавиша с лампочкой). Щелкаем, отыскиваем в подменю лампочку, обозначаем помещение, например «гостиная» и, удерживая левую кнопку мышки, перетаскиваем лампочку в нужное место (рис. 1.8).

Рис. 1.8. Размещение подсистемы управления светом

После расстановки источников света (люстр, бра, групп светильников), задаем нужные свойства и меняем, если это нужно, обозначения. При использовании диммеров следует обратить внимание на опцию **Ramp to level gradually** (переходить на уровень плавно) слева в рубрике **On/Off Rate** (ход включения/выключения). Если не установить эту опцию, плавное включение будет в программе недоступно. Остальные опции позволяют выбрать скорость нарастания яркости – **Dim rate** (ход выключения), установить уровень яркости в ночное и дневное время, яркость свечения индикаторов в ночное и дневное время и т.д. Задание свойств выключателей подразумевает, что загрузка основной программы в процессор системы сопровождается передачей всех настроек соответствующим модулям.

После идентификации источника света на первой вкладке **Control** (управление) можно управлять им в интерактивном режиме, когда компьютер подключен к системе. На других вкладках можно построить сцены управления светом (вместе с люстрой, щелкнув кнопку ее включения, можно включать потолочную подсветку в спальне и т.п.). Включается меню настроек (свойств – properties) двойным щелчком левой кнопкой мыши или щелчком правой кнопкой при позиционировании на объекте и выборе в раскрывающемся меню раздела свойства (properties) – рис. 1.9.

Расставив источники света и задав все необходимые свойства, пора заглянуть в раздел основной инструментальной панели **Program**. В данном случае я раскрыл подменю спальни (bedroom) – рис. 1.10.

В левом окне вкладки событий (events) в спальне обозначены все светильники по их функциональному назначению. Обязательно ли это? Нет. Для профессионалов, возможно, даже удобнее иной подход с обозначением помещений 1, 2, 3, а источников света – 1 Light1, 1 Light 2 и т.д., поскольку в рабочей документации будут таблицы соответствия. Однако я считаю удобным сразу обозначать функциональное назначение помещений и оборудования. В дальнейшем это позволит осмысленнее подходить к программированию событий – события в кухне и спальне, мне кажется, могут сильно различаться.

Рис. 1.9. Задание свойств модулей управления светом

Рис. 1.10. Окно программирования системы Landmark

Аналогично в проект добавляются системные устройства управления, аудио- и видеооборудование, датчики и т.п. На этом подготовка к работе с проектом закончена, можно начинать программирование.

Отвлечемся немного от технических деталей. Поскольку все системы, с которыми мне приходилось сталкиваться, в равной мере позволяют осуществить, хотя бы формально, любую программу, самое время определить, а что мы хотим в ней видеть?

Посмотрим, насколько просто реализовать в системе Landmark сценарий, приведенный в предисловии под названием «Возвращаюсь я с работы». Я несколько изменю предыдущий вариант конфигурации проекта, оставив прихожую, кабинет, гостиную и кухню.

Расставим необходимые элементы (рис. 1.11).

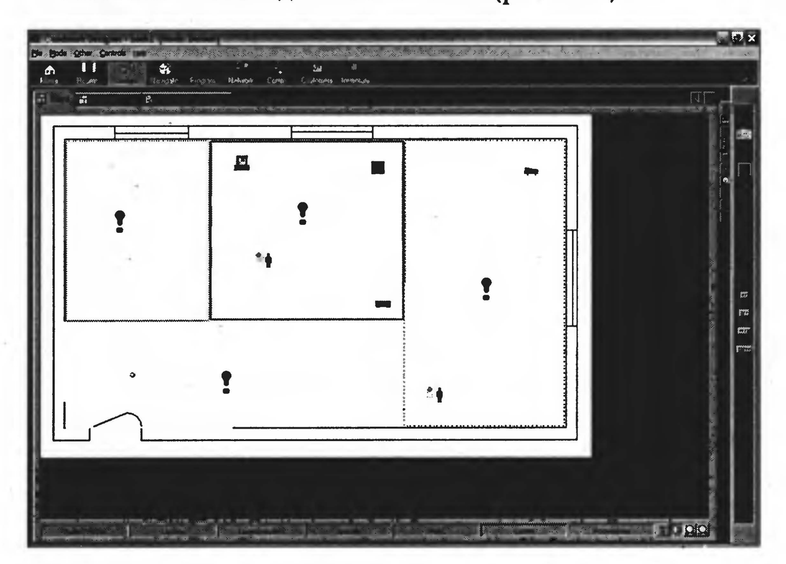


Рис. 1.11. Вид проекта в системе Landmark для сценария «Возвращаюсь я с работы»

Во всех комнатах я устанавливаю свет. Везде, кроме кабинета, добавляю датчики движения, в гостиной – телевизор и сенсорную панель управления, а на кухне – плиту. В гостиной я поместил и **CardFrame** (конструктив, в котором расположатся модули). По команде «спецификации» оборудования

(Spec) программа заполнит **CardFrame** необходимыми модулями (рис. 1.12).

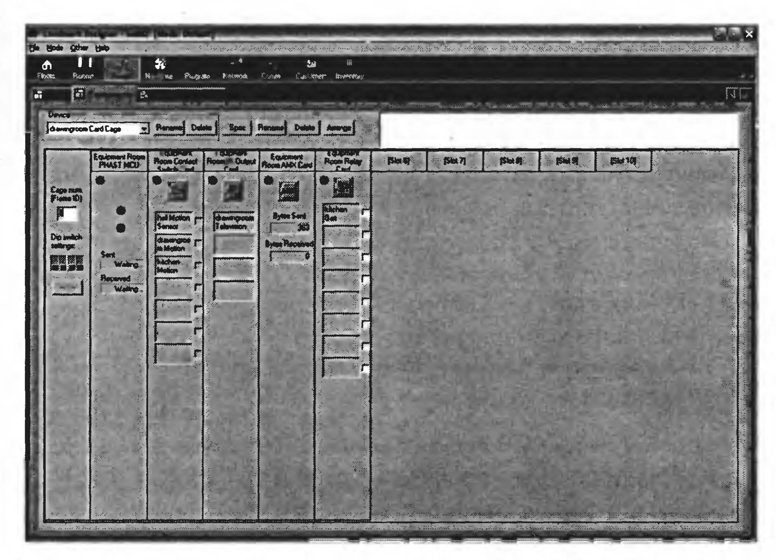


Рис. 1.12. Размещение модулей в системе Landmark

Теперь можно перейти к программированию.

Опишем события: я прихожу домой, сработавший датчик движения в холле зажигает свет, включает электрочайник, чтобы, пока я переоденусь, вода для кофе вскипела. Конечно, чайник я наполняю водой и ставлю на подставку перед уходом на работу (рис. 1.13).

Переодевшись, я перехожу в кухню, наливаю кофе и иду в гостиную. Программа выключает свет в холле, электрочайник на кухне и включает телевизор в гостиной, едва датчик движения в гостиной отмечает мое появление (рис. 1.14).

Модули, которые программа задействовала самостоятельно, – это модуль процессора, модуль, поддерживающий сенсорную панель, модуль цифрового ввода. К последнему мы подключим датчики движения. Еще система добавила релейный модуль. С помощью этого универсального модуля можно включить и выключить такой бытовой прибор, как электрический чайник (выключается он сам, но модуль дополнительно отключит его и от сети). Если параметры контактов реле



Рис. 1.13. Начинаем программировать сценарий «Возвращаюсь я с работы»

Рис. 1.14. Завершаем программировать сценарий «Возвращение домой»

не позволяют управлять включением мощной нагрузки, можно применить пускатели, и реле будет включать и выключать пускатель. Модуль трансляции ИК-кодов займется управлением телевизора. С помощью сенсорной панели я могу легко управлять всеми бытовыми устройствами квартиры, например открыть входную дверь и включить свет в холле, если сосед (или соседка) зашел в гости на чашку кофе.

Система Landmark централизованная. Она может работать под управлением компьютера, собственного процессора, в который загружается программа, и переходить от управления с помощью компьютера к управлению от собственного процессора, если компьютер по какой-то причине выключается.

Конечно, она имеет средства организации распределения аудио- и видеосигналов, охраны дома, управления климатическими устройствами и т.д. Под «и т.д.» я подразумеваю дальнейшее описание работы с системой Landmark, что, однако, в данном случае не является целью повествования.

А вот как среда программирования выглядит в NetLinx Studio – программе, предназначенной для работы с процессорами и модулями корпорации AMX. После запуска программы можно создать новый проект обычными для Windows средствами (**Меню ➡ Файл ➡ Новый**), или открыть существующий (рис. 1.15).

Язык программирования больше похож на процедурно-ориентированный язык, что отражается и на среде программирования, хотя не следует забывать, что он остается специализированным. Встроенный редактор позволит скопировать программу в блокнот из Windows и продолжить программирование в блокноте. При наличии готовых программных блоков они легко добавляются в программу. Таким образом, даже достаточно сложную программу можно быстро скомпоновать из рабочего материала предыдущих разработок. Если вы специализируетесь на работе с системой NetLinx, то кроме программных блоков, предлагаемых производителем, у вас много своих, надеюсь, систематизированных, проверенных и отлаженных. Вот с установкой устройств проблем несколько

Рис. 1.15. Система «Умный дом» корпорации AMX

больше. После начального процесса установки устройств программа будет выглядеть, примерно, как показано на рис. 1.16.

Рис. 1.16. Установка оборудования в системе корпорации AMX

В обеих системах после начального подготовительного процесса, который никак нельзя назвать сложным, мы готовы приступить к программированию.

Система StarGate – X10

Системы, работающие по протоколу X10, – это другой ценовой полюс. Множество производителей выпускают как отдельные компоненты, так и законченные системы автоматизации. Можно использовать централизованное построение или создать децентрализованную систему. В качестве примера рассмотрим систему StarGate (JDS).

Как и предыдущая, она позволяет быстро объединить события с реакцией системы, создав программу обслуживания дома. Имеет система и схожий набор устройств, в котором, как и в других системах, есть все необходимое для создания подсистем управления светом, аудио- и видеооборудованием, климатической и охранной подсистемы, подсистемы управления, например, садовым оборудованием или гаражом.

О перечне модулей, составляющих физическое наполнение системы, можно судить по меню, которое открывается в программе после обращения к разделу **Define** основного меню (рис. 1.17).

При создании нового проекта система добавляет начало и конец программы (рис. 1.18).

После запуска программы и выбора **New Schedule** (через основное меню **File** или инструментальную панель – первая кнопка) в оглавлении мы готовы перечислить оборудование нашего проекта. Как и в случае с системой Landmark, начнем со света.

В основном меню выберем **Define ➡ X10 Device...**

Открывается таблица, в которой можно обозначить все источники света (но не только). В системах, работающих с сетевым протоколом X10, адрес устройств состоит из двух знаков: латинской буквы от A до P и цифры от 1 до 16. Всего, таким образом, можно адресоваться к 256 устройствам, что достаточно для довольно большой системы. Распишем световую подсистему (рис. 1.19).

Расположение выключателей света можно задать в описании – **Location**, назначение – **Description**.

Рис. 1.17. Меню устанавливаемого оборудования в системе StarGate

Рис. 1.18. Создание проекта в системе StarGate

Я хочу повторить предыдущий небольшой сценарий «Возвращаюсь я с работы», реализованный в системе Landmark.

Рис. 1.19. Расстановка адресуемых модулей управления светом

Поэтому выберу светильники и добавлю релейный модуль для включения электрического чайника (рис. 1.20).

Рис. 1.20. Добавление оборудования для сценария «Возвращаюсь я с работы»

Осталось добавить в проект датчики движения, которые будут работать со встроенным модулем цифрового ввода (**Define ➡ IR and IO Devices**). Я выбираю модуль ввода-вывода, щелкаю кнопку **Define** и определяю свои датчики движения (рис. 1.21).

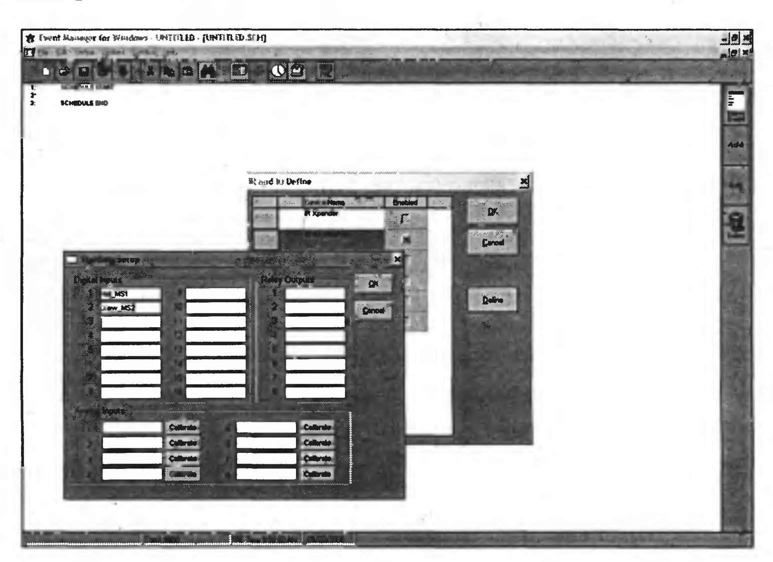


Рис. 1.21. Добавление датчиков для сценария «Возвращаюсь я с работы»

Сохраним выбор, и можно начинать программирование.

В редакторе выбираем кнопку **New Event**, обозначаем первое событие как `come_home` и к строке `if` с помощью кнопки **Add** и выбора из меню **Digital Input** выбираем событие – датчик движения `hall_MS1` перешел в состояние `ON` (рис. 1.22).

Продолжая этот процесс, с помощью кнопок **New Event** и **Add** пишем программу (рис. 1.23).

Обе системы – и Landmark, и StarGate – после загрузки программы в центральное управляющее устройство встретят меня после работы, зажгут свет в прихожей, вскипятят воду для кофе. А когда я перейду в гостиную, включат телевизор, чтобы я, усевшись в любимое кресло, посмотрел, что произошло в мире за день.

Рис. 1.22. Начинаем программирование сценария в системе StarGate

Рис. 1.23. Завершаем программирование сценария в системе StarGate

К сказанному выше следует добавить, что для управления телевизором обе системы требуют дополнительного устройства, которое можно назвать модулем считывания ИК-команд. Они предназначены для запоминания ИК-кодов с пультов управления, чтобы эти коды можно было впоследствии воспроизвести системным устройством. В системе Landmark это дополнительное устройство носит название IRIS. А StarGate работает с IR Xpander. Последнее устройство не только прочитывает ИК-команды с пультов и запоминает их, но может распознавать эти команды, оно же воспроизводит их.

Задание адреса устройств системы StarGate может производиться либо переключателями, установленными на устройстве – один переключает буквы от А до Р, второй цифры от 1 до 16 – либо программно. В последнем случае, как правило, требуется помощь компьютера или одного из контроллеров X10.

Беглый обзор двух систем «Умный дом» завершен. Зачем он понадобился?

Да чтобы можно было составить представление о том, как это выглядит в профессиональных разработках, иметь цель и сравнить сделанное нами с профессиональными разработками. Затем, чтобы вы могли при желании продолжить проект, описанный в книге, до уровня профессиональной разработки.

Но не следует забывать – за видимой простотой процесса программирования как в Landmark, так и в StarGate скрывается система, которая рано или поздно проявит все свои свойства. При первом знакомстве может возникнуть иллюзия, что программирование по легкости схоже с игрой в кубики: сложил так – получил домик, переложил иначе – получаешь паровозик. Подобная иллюзия может стать причиной того, что однажды... домик запыхтит и поедет...

И последнее. Что следует «подвергать» автоматизации. Вот, как на это смотрят специалисты PHAST.

Что должно автоматизироваться

В принципе, любое электрическое или механическое устройство, любая подсистема внутри или снаружи вашего дома может быть автоматизирована, хотя бы ненамного.

Поскольку количество устройств и подсистем, которые вы можете интегрировать с системой домашней автоматизации, огромно, важно выбрать систему достаточно гибкую, расширяемую, обновляемую и оптимальную по стоимости. Это должна быть система, которая легко устанавливается, программируется и имеет программный интерфейс, не требующий освоения новой техники программирования каждый раз, когда вы добавляете в систему новое устройство.

Корпорация PHAST развивает современную автоматизацию дома, которая полностью отвечает этим требованиям.

Заметьте: так как список характеристик, относящихся к разным системам автоматизации и компонентам, которые могут быть автоматизированы, велик, система автоматизации дома может стать весьма сложной. Для преодоления проблемы вам следует обращаться к системам, с которыми вы лучше всего знакомы. PHAST настоятельно рекомендует тщательное изучение всех незнакомых систем или устройств, прежде чем вы примете решение о внедрении их в систему Landmark.

«Умный дом» вашего производства

В первую очередь рассмотрим, как могут выглядеть решения, о которых упоминается в предисловии.

Темный холл

Сценарий решения: работая за компьютером дотемна, чтобы выйти в темный холл, не натыкаясь на мебель, я с компьютера включаю бра в холле.

Для реализации этого решения мне нужен релейный модуль, имеющий одно реле, и простая основная программа на компьютере, которая отправляет команды «включить свет» и «выключить свет», адресованные этому модулю (рис. 1.24).

Аналогичное решение можно применить с небольшой модификацией (можно и со значительной) и для случая, когда ночью приходится вставать, чтобы зайти на кухню попить или выйти в туалет. Ночной светильник в холле избавит вас от опасности споткнуться по дороге, но не будет резать глаза после темноты в спальне. Модификация в данном случае

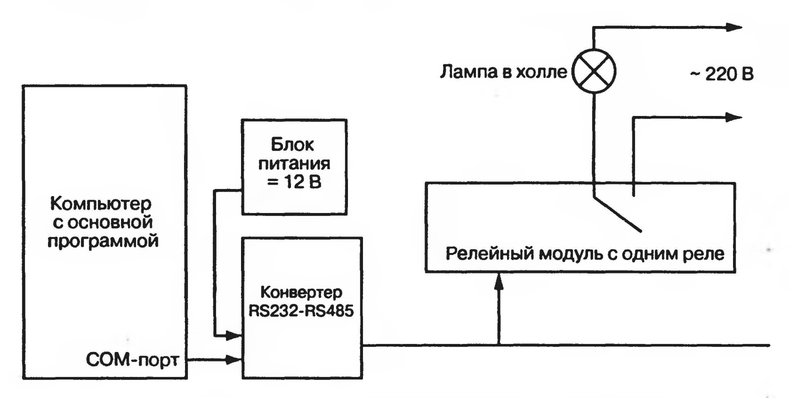


Рис. 1.24. Структурная схема системы для сценария «Темный холл»

относится к тому, чтобы в программе, заложенной в контроллер релейного модуля, один из выводов порта предназначался к присоединению кнопки. Эту кнопку (клавишу, выключатель) вы можете расположить возле прикроватной тумбочки или на ней. Нажатием кнопки вы включаете и выключаете реле без участия компьютера. Кнопок можно сделать несколько, если вы в доме не один.

Когда я говорил о значительной модификации, то имел в виду, что кнопку возле кровати можно снабдить радиопередатчиком, а релейный модуль – радиоприемником. В этом случае вам не понадобится добавлять провода в квартире.

Возвращаюсь я с работы

Сценарий решения: по возвращении домой я открываю входную дверь. Система зажигает свет в прихожей, кипятит воду для кофе. Когда я перехожу в гостиную с чашкой кофе, она включает телевизор, выключает свет на кухне и в прихожей (рис. 1.25).

Для реализации этого решения мне потребуется модуль цифровых вводов, к которому подключаются герконовые датчики – первый, установленный на входной двери, и второй – на двери в гостиную. Мне понадобится релейный модуль с тремя реле, одно из которых включает свет в прихожей, второе – свет на кухне, третье – электрочайник. Задействован будет также модуль излучения ИК-команд с излучающим светодиодом, расположенным возле телевизора. Основная

программа на компьютере будет иметь программные блоки, обслуживающие системные модули.

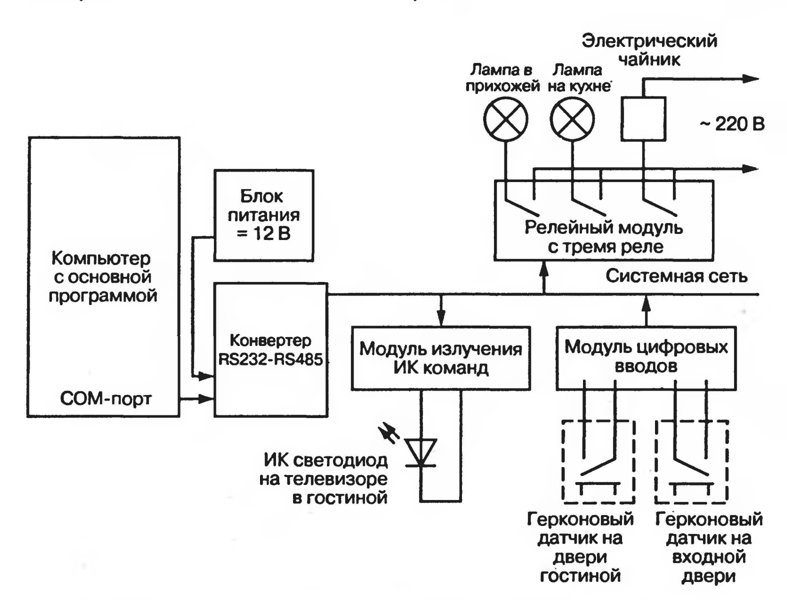


Рис. 1.25. Структурная схема системы для сценария «Возвращаюсь я с работы»

Основным событием для программы, работающей на компьютере, в данном решении становится изменение состояния герконового датчика на входной двери. Это событие инициирует включение света в прихожей и на кухне, включение электрочайника (выключается он сам, когда вода закипит). Следующим событием становится срабатывание герконового датчика не двери гостиной. По этому событию программа выключает свет в прихожей, на кухне, выключает электрочайник (не обязательно). Затем через модуль излучения ИК-команд отправляется код включения телевизора (многие телевизоры включаются командой выбора конкретного канала) на выбранную программу.

Возвращаюсь я с работы (модификация решения)

Сценарий решения остается прежним (рис. 1.26).

Мне не нравится держать компьютер, на котором работает основная программа, весь день включенным только для

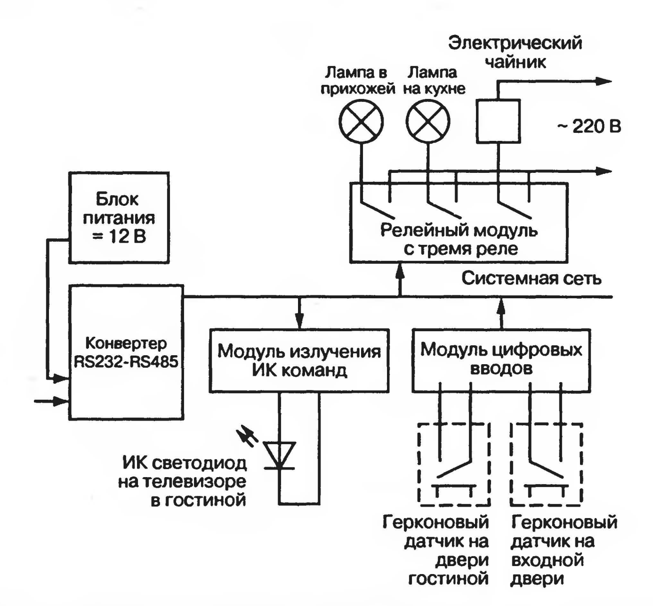


Рис. 1.26. Структурная схема системы для модификации предыдущего сценария

того, чтобы вечером он управлял работой системы. Я модифицирую модуль цифровых вводов. Теперь при срабатывании герконового датчика входной двери он сам отправляет команды включения релейному модулю. А при срабатывании датчика на двери гостиной он отправит команды выключения релейному модулю и команду включения телевизора модулю излучения ИК-команд.

После модификации постоянно включенным оказывается только маломощный блок питания, обеспечивающий модули постоянным напряжением 12 В, и сами модули, потребляющие очень маленький ток.

Если мне не хочется по какой-то причине устанавливать герконовый датчик на двери гостиной, я могу использовать клавишу пульта, закрепленного на стене в прихожей рядом с выключателем света. Проходя мимо пульта, я нажимаю эту клавишу, что инициирует продолжение работы того же сценария. Вместо герконового датчика на двери в гостиную можно использовать датчик движения, установленный в гостиной или замок с цифровым кодом на входной двери, который подаст команду

«встретить хозяина». Никаких переделок в системе для выполнения сценария мне делать не нужно. И, естественно, для подобного решения не нужен конвертер RS232–RS485 (как и в последующих решениях), но подразумевается, что компьютер вы все-таки используете, например, для выполнения первого или других сценариев, которые придумаете или сочтете интересными. Поэтому конвертер остался в системе.

Создание эффекта присутствия

Практически все системы автоматизации жилья позволяют реализовать подсистему охраны. В отличие от специализированных систем охраны, основным достоинством которых кроме высокой надежности является возможность стандартного подключения к централизованным пультам охранных ведомств, системы автоматизации жилья позволяют создать эффект присутствия. Посмотрим, как может выглядеть сценарий подобного решения, например, на вашей даче зимой, когда вы там не живете.

Сценарий решения: рано утром на террасе включается свет, который спустя несколько минут гаснет. Следом включается свет в одной из комнат. Включается радиоприемник. Через полчаса он выключается. В комнате слышатся голоса, зажигается свет в другой комнате. Через час в доме наступает тишина. Но ненадолго. Через час или два в доме вновь слышны голоса. Включается и выключается свет. Включается и выключается радиоприемник (рис. 1.27).

Для реализации решения мне понадобится модифицированный релейный модуль. Я использую транзисторный радиоприемник и кассетный магнитофон с пленкой, на которой записаны обычные домашние шумы и разговор. В данном случае сеть используется, если управление светом удобнее разнести по дому. В такой модификации один из релейных модулей будет модифицирован, остальные останутся универсальными.

Модификация релейного модуля №1 сведется к тому, что будут использованы таймеры для задания интервалов времени, которые становятся событиями системы, а отклик системы на события (сигналы таймеров) инициируется модифицированным релейным модулем, который отправляет команды

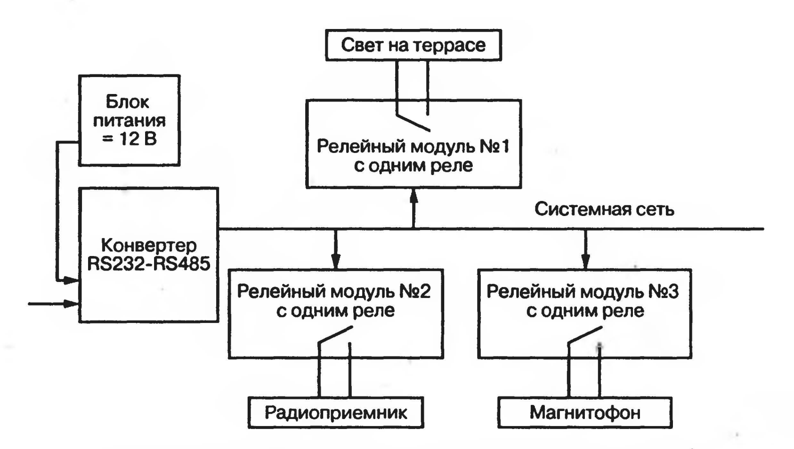


Рис. 1.27. Структурная схема системы для сценария «Эффект присутствия»

включения и выключения релейным модулям №2 и №3. Контакты реле, конечно, разрывают питающее напряжение. Если вы действительно не живете на даче, но хотите использовать подобную подсистему, для имитации присутствия можно подумать, например, о применении вместо силового освещения низковольтных маломощных светильников, работающих от батареек. Можно добавить сигнализацию, добавив модуль цифровых вводов, к входам которого подключить герконовые датчики, реагирующие на открывание дверей и окон. А в качестве устройства тревоги использовать сирену. Бесспорно, вовсе не обязательно для реализации сценария задействовать систему. Но, если вы будете использовать систему, то в летнее время, когда вы живете на даче, можете возложить на нее обязанности ежедневного полива огорода, включения света на крыльце в ночное время и т.д.

К одному из внешних проявлений гибкости систем «Умный дом» я бы отнес тот факт, что трудно говорить о каких-либо решениях без привязки к конкретным объектам, к конкретным условиям, и даже к конкретным людям.

Если вы дочитаете книгу до конца, пусть и не последовательно, а выборочно, то сможете собрать все необходимые модули, модифицировать их и соединить. Обойдется это рублей в 200–500 (без учета стоимости сирены, радиоприемника и магнитофона). Вы проверите работу системы на столе и убедитесь в ее работоспособности. В один из выходных

дней вы поедете на дачу и установите систему на месте. Проверите ее в работе. Убедитесь в том, что она работает.

Осталось убедиться в том, что система нужна вам именно в том виде, в каком описана в сценарии, включая сирену, срабатывающую при открывании дверей и окон.

Если ваш дачный участок охраняется, то да. Имитация присутствия может отпугнуть нежелательных визитеров, а сирена, если они все-таки решат нанести визит, не только отпугнет их, но и привлечет внимание сторожа.

Если дачный участок не охраняется, то нет. Имитация присутствия отпугнет тех, кто будет наблюдать за вашей дачей длительное время, прежде чем отважится на посещение. Да и то в том случае, если не выпадет снег. Люди в доме, не оставляющие следов на снегу, скорее убедят наблюдателя в том, что его пытаются обмануть. Сирена, которая воет на весь поселок без появления людей, «поднятых по тревоге», тоже не слишком убедительна.

Если дверь, на которую вы установили герконовый датчик для защиты от вторжения, не ходит ходуном под ветром, то да. В противном случае, нет, даже, если дачный поселок охранят сторож. При перемещениях двери датчик будет срабатывать, а сторож, привыкший к ложным срабатываниям системы, перестанет обращать внимание на сирену.

Да и саму систему, если поселок охраняется, следует, на мой взгляд, сильно видоизменить. Модули цифровых вводов и сирену поставить в каждый дом поселка, соединив их между собой, а лампу тревоги (или любой индикатор) разместить в доме сторожа. Если сторож время от времени будет оставлять «следы на снегу», то эффект присутствия окажется полезен.

Как видно из вышесказанного, конкретное решение следует выбирать только с учетом всех обстоятельств и особенностей применения.

Это же касается и, например, реализации описанных модулей в квартире. Если вы достаточно опытны в работах с электричеством, можно эксперимент довести до конца, подключив к релейному модулю (или модулю с триаком) торшер или настольную лампу. Если нет, лучше остановиться на применении светодиода. Тем более что сейчас появились светодиоды, обладающие очень бо́льшой светоотдачей. Их света вполне может хватить для подсветки темного коридора или холла.

Цель проекта

Цель проекта – разработка любительской системы автоматизации жилья. За основу возьмем системы, о которых говорилось выше. Если не вдаваться в тонкости реализации разных концепций, на первом этапе будущую систему можно представить в виде центрального управляющего устройства и набора модулей, выполняющих разные функции, но подчиненных одной задаче – следить за состоянием датчиков и устройств управления, чтобы на основе их состояния включать, выключать и переключать бытовую технику (рис. 1.28).

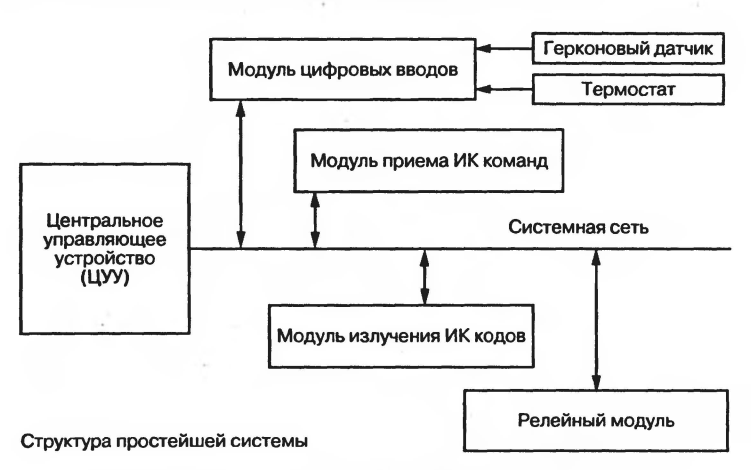


Рис. 1.28. Структура простейшей системы с базовыми модулями

В качестве средств управления в системе промышленного производства используются сенсорные панели и универсальные ИК-пульты с запоминанием кодов.

Не готов утверждать, что любительская разработка подобного рода устройств управления столкнется с непреодолимыми трудностями, но если серийно производимое устройство оценивается в продаже в тысячу (и несколько тысяч) долларов, то и в любительской разработке оно может стоить не дешевле. По этой причине разработку средств управления подобного типа лучше пока оставить за профессионалами. Мы постараемся реализовать простую систему, в которой компьютер будет

играть роль центрального управляющего устройства (и, в какой-то мере, устройства управления), и которая будет иметь несколько базовых модулей: релейный модуль, модуль приема системных ИК-команд, модуль излучения ИК-кодов и модуль цифровых вводов.

Каково назначение каждого из этих модулей?

Релейный модуль. Получая команды центрального управляющего устройства, он включает и выключает соответствующее реле. С помощью контактов реле можно включать и выключать настольную лампу, торшер (и свет в комнате, установив модуль на место обычного выключателя, но я не советую делать это, если вы не профессиональный электрик), телевизор или музыкальный центр. Контактами реле может включаться и выключаться электрический чайник и утюг (возможно, понадобится добавить более мощный контактор). С его же помощью можно «перемещать музыку», подключая к музыкальному центру громкоговорители, установленные в разных помещениях. Одним словом, с помощью контактов реле можно включать и выключать все, что можно включать и выключать в принципе.

Модуль приема системных ИК-команд. Системой хочется покомандовать. И не только с компьютера. Используем старый пульт управления от видеомагнитофона или телевизора, который завалялся на полке, и применим его коды для управления системой.

Модуль излучения ИК-кодов. Чтобы управлять с помощью системы телевизором или видеомагнитофоном необходимо иметь устройство, которое излучает ИК-коды управления ими.

Модуль цифровых вводов – будет соединяться с датчиками, например, герконовыми и по запросу центрального управляющего устройства будет передавать состояние датчиков.

В качестве сетевого интерфейса (дверка, за которой начинается дорога ко всем модулям системы) используем двухпроводный интерфейс RS485. Длина линии может достигать 1000 м, все системные устройства включаются параллельно, линия мало подвержена влиянию наводок и сама не наводит шумов на другие линии. Если для экспериментов применить четырехжильный провод, то по двум оставшимся проводам

можно передавать напряжение питания для всех системных устройств. Этого достаточно для наших целей. Тем более что в плане деталей, которые нужны для создания интерфейса, это одна микросхема и один резистор.

Какие схемы нам потребуется собрать до начала работы?

Программатор, работающий с программой PonyProg2000 (упрощенная схема, более полная схема для PIC контроллеров будет приведена в Приложении), показан на рис. 1.29.

В оригинальной версии есть примечание – микросхему стабилизатора напряжения LM2936–Z5 не советуют менять на стабилизаторы серии 7805. Поскольку рекомендуемая микросхема есть в продаже, я не стал пренебрегать советом, хотя, по отзывам многих, собиравших программатор, этим советом можно пренебречь. Список необходимых для сборки программатора составляющих приведен в табл. 1.1.

Таблица 1.1. Спецификация программатора

№	Обозначение	Изделие	Количество	Цена (р.)	Примечания
1	U1	Панелька DIP18	1	21	
2	U2	LM2936-Z5	1	68	Не использовать 7805
3	Q2, Q3	КТ315Д	2	10	
4	Q1	КТ361Д	1	5	
5	D1–D3	КД522	3	3	
6	Z1–Z3	КС147	3	6	
7	Z4	КС213	1	3	
8	R1	10 кОм 0,25 Вт	1	0.5	
9	R3, R8–R10	4,7 кОм 0,25 Вт	4	2	
10	R4	100 кОм 0,25 Вт	1	0.5	
11	R2, R7	1 кОм 0,25 Вт	2	1	
12	R5	2,2 кОм 0,25 Вт	1	0.5	
13	C1, C2, C5	100 нФ	3	30	
14	C3	1 нФ	1	5	
15	C4, C6	47 мкФ 16 В	2	20	
16	J1 (DB9)	Разъем, гнездо	1	10	

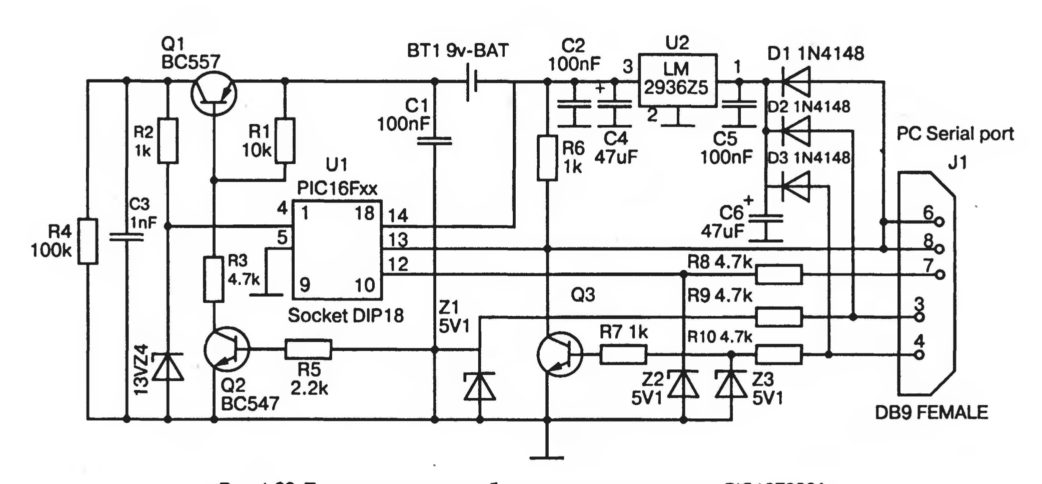


Рис. 1.29. Программатор для работы с микроконтроллером PIC16F628A

Ориентировочная стоимость элементов – 186 руб., макетная плата 100 руб. Всего 286 руб.

Программатор подключается к COM-порту (я подключал его к COM2, отведя COM1 для конвертера). При точной и аккуратной сборке схема работает сразу.

В качестве соединительного кабеля между программатором и компьютером я использовал несколько проводов плоского кабеля, которые были «под рукой». Думаю, подойдет кабель от старой «мышки» или отрезок с витыми парами для компьютерной сети. Длину этого отрезка лучше сделать небольшой, чтобы с программатором было удобно работать.

Если купить макетную плату достаточных размеров, часть ее можно использовать для программатора, а часть пригодится для конвертера RS232–RS485. Остатка же хватит на макетную плату для прототипов всех модулей. У меня программатор свободно разместился на плате 90×40 мм.

Конвертер RS232–RS485 – нужен для общения компьютера со всеми модулями системы. Он также несложен в сборке, поскольку представляет собой пару микросхем с небольшим количеством дополнительных элементов. Я использовал только один резистор 120 Ом в линии. При длинной линии (в сотни метров) такой же резистор желательно поставить на конце линии, выполнив ее витой парой (рис. 1.30).

Необходимые элементы конвертера приведены в табл. 1.2.

Таблица 1.2. Спецификация конвертера

№	Обозначение	Изделие	Количество	Цена (р.)	Примечания
1	DA1	MAX232ACPE	1	80	
2	DA2	MAX1483	1	96	
3	R1	120 Ом 0,25 Вт	1	1	
4	C1–C5	0,1 мкФ	5	5	
5	DA3	LM78L05	1	68	
6	C6	47 мкФ	1	20	
7	DB9	Разъем гнездо	1	10	
8	X1–X2	Клеммник	1	10	Любой на 6 конт.

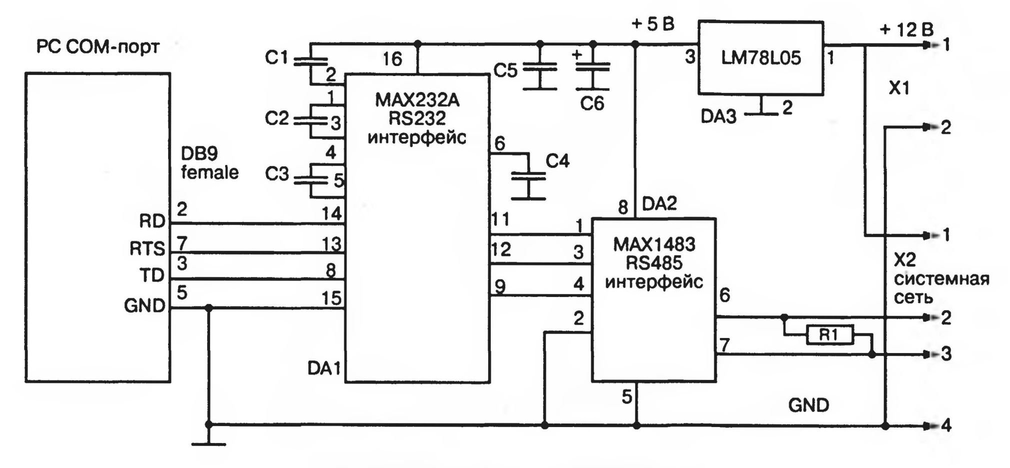


Рис. 1.30. Схема конвертера RS232–RS485

Ориентировочная стоимость изделий – 290 руб.

Клеммник я использовал для подключения линии, внешнего (и общего для всех модулей) источника питания 12 В и проводов передачи питания на макетную плату, куда установлен 4-контактный клеммник. Таким же образом можно подключать все модули, соединяя их последовательно. При этом следует учитывать, что по мере удаления от общего источника питания напряжение может понижаться за счет падения напряжения на проводах. Это сказывается на работе удаленных релейных модулей. Реле с рабочим напряжением 12 В может потреблять ток порядка 50 мА. Обе микросхемы конвертера подключены к источнику питания 12 В через микросхему стабилизатора 7805 так же, как микросхемы модулей. Некоторые параметры микросхемы MAX1483 и цоколевка показаны на рис. 1.31 и 1.32.

MAX3082/MAX3085/MAX3088

TRANSMITTING				
INPUTS			OUTPUTS	
$\overline{RE}$	DE	D1	B/Z	A/Y
X	1	1	0	1
X	1	0	1	0
0	0	X	High-Z	High-Z
1	0	X	Shutdown	

RESEIVING			
INPUTS			OUTPUT
$\overline{RE}$	DE	A-B	RO
0	X	≥ 0.05V	1
0	X	≤ ? 0.02V	0
0	X	Open/shorted	1
1	1	X	High-Z
1	0	X	Shutdown

Рис. 1.31. Таблица состояний микросхемы MAX1483

На этом приготовления к началу работы можно считать законченными, если вы установили на компьютере программы MPLAB и PonyProg2000.

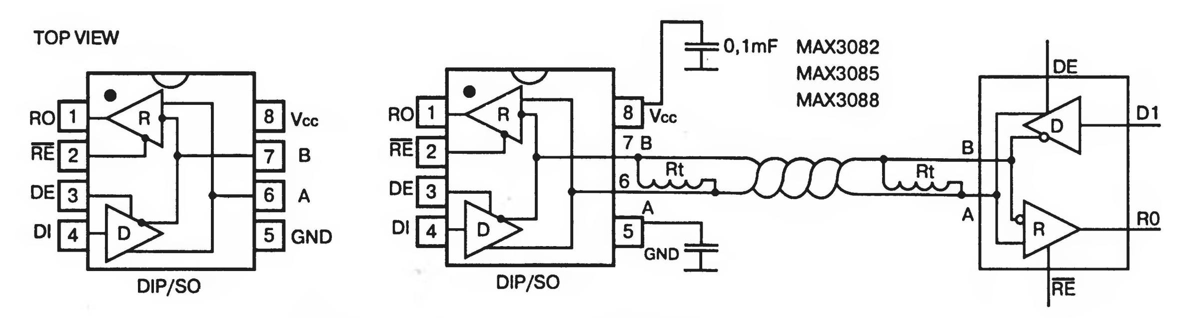


Рис. 1.32. Типовая схема включения микросхем MAX3082 (MAX1483)

Первую разработку проведем «на бумаге», точнее в редакторе программы MPLAB. После программирования контроллера «на бумаге» перейдем к его налаживанию в программе MPLAB.

Схема и программа релейного модуля

Функциональная схема модуля состоит из интерфейса, контроллера и адресного селектора, образующих базу для построения остальных модулей, а отличительной особенностью данного модуля является использование реле (рис. 1.33). Реле я включил через транзисторный ключ. В зависимости от конкретного реле, которое вы выберете, транзисторный ключ может оказаться лишним.

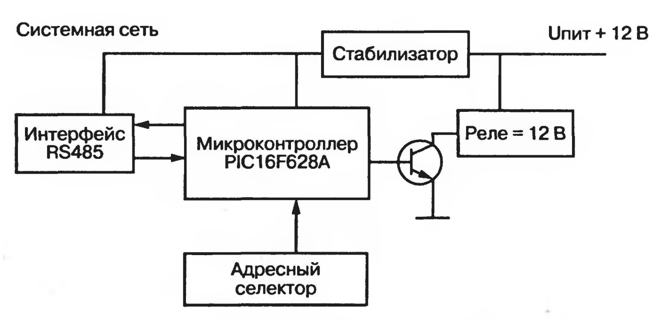


Рис. 1.33. Функциональная схема релейного модуля

Я не стал устанавливать реле на макетной плате. Реле – элемент достаточно дорогой, и нет нужды покупать его без твердого намерения использовать. Вместо него к выходам микроконтроллера были подключены красные светодиоды АЛ307. Как выяснилось позже, получилось очень полезное решение при наладке и разработке других модулей (рис. 1.34).

При сборке макета потребуются не все элементы, обозначенные в табл. 1.3.

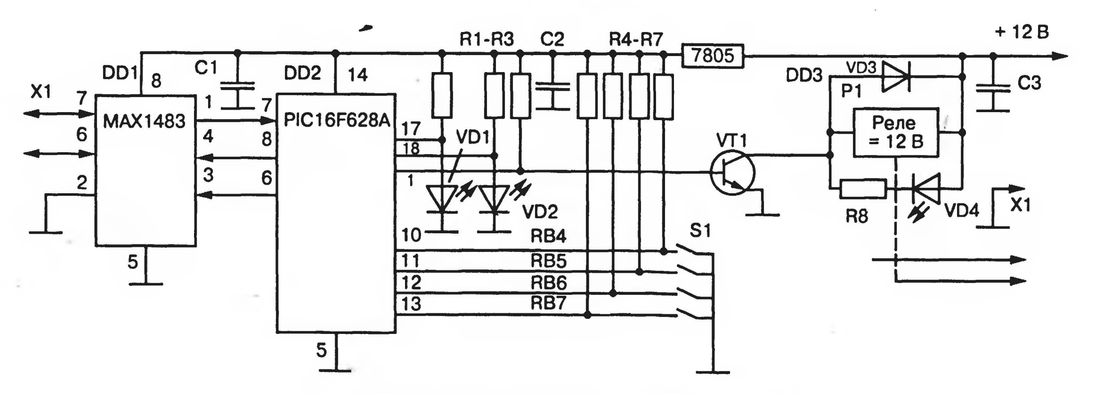


Рис. 1.34. Принципиальная схема релейного модуля

Таблица 1.3. Спецификация релейного модуля

№	Обозна-чение	Изделие	Количество	Цена (р.)	Примечания
1	DD1	MAX1483	1	96	
2	DD2	PIC16F628A	1	100	Установить на панельку
3	DD3	LM2936-Z5	1	68	
4	VT1	КТ503Г	1	5	
5	VD1, VD2	АЛ307	2	3	
6	VD3	КД522	1	3	
7	VD4	АЛ307	1	3	
8	P1	12В, 5А/~220 В	1	170	
10	R1–R3	1 кОм 0,25 Вт	3	1	
11	R4–R7	10 кОм 0,25 Вт	4	1	
12	R8	2,2 кОм 0,25 Вт	1	1	
13	C1, C2	0,1 мкФ	2	5	
14	C3	100 мкФ 16 В	1	20	
15	X1	Кл. 6 конт.	1	10	
16	S1	Пер. 2 пол. 4 нап.	1	50	
17	SOC	DIP18	1	21	

Ориентировочная стоимость элементов – 558 руб.

В целях экономии я отказался от установки на макетную плату реле и переключателя для организации адресного селектора, распаяв соответствующие выводы, чтобы получить один адрес. На схеме показано одно реле, но их количество можно увеличить до 7–8, используя все свободные выводы портов A и B. Это определяется конкретными соображениями по применению модуля. Например, если вы планируете использовать релейный модуль в своей комнате для включения торшера или настольной лампы, достаточно одного реле. Для безопасного включения лампы я советую использовать закрытую розетку со стандартным сетевым проводом. Одна из жил этого провода должна разрываться контактами реле. Настольная лампа включается в розетку, которая, в свою очередь, подсоединяется к сети ~220В. При замыкании контактов реле напряжение подается на лампу. Поскольку неизвестно, какой из проводов вы коммутируете: нулевой или фазный, все работы лучше производить без напряжения.

После тщательной изоляции розетки и всех соединений включите лампу в вашу розетку, подключите мультиметр в режиме измерения сопротивления к вилке вашего провода и включите реле. Мультиметр должен показать такое же сопротивление, что и настольная лампа, если измерить ее сопротивление. После команды выключения реле тестер должен показать обрыв. В любом случае при первом включении следует соблюдать осторожность, а релейный модуль сразу поместить в любую подходящую пластмассовую коробку.

Команды, на которые модуль должен реагировать:

- включить – Rxx$xN;
- выключить – Rxx$xF;
- передать статус – Rxx$xS.

При передаче статуса используем следующую символику:

- включено – Rxx#xN;
- выключено – Rxx#xF.

Здесь Rxx$xN означает: R – релейный модуль, xx – два символа адреса от 00 до 15, $ – символ команды (# – символ статуса), x – номер реле от 0 до 7, N – включить (F – выключить).

Набор и формат команд вы можете переопределить по своему вкусу или сообразуясь с целями. Для этого достаточно внести изменения в исходные коды, приводимые ниже.

Для адресации используются четыре бита порта B (RB4–RB7), что позволяет адресоваться к 16 модулям.

Для тех, кому не терпится опробовать готовые решения, кому это не интересно или не нужно знать, как программировать модуль, я сразу приведу текст программы модуля на ассемблере, текст программы на языке программирования C и HEX-файл для загрузки в программатор.

Программа модуля на ассемблере

```
   list   p=16f628a
   #include p16f628a.inc

; Инициализация модуля.

BCF STATUS, RP1    ; Выбор банка 0.
```

```
BCF STATUS, RP0
CLRF PORTA          ; Настройка порта A.
MOVLW 0x07
MOVWF CMCON
BCF STATUS, RP1     ; Выбор банка 1.
BSF STATUS, RP0
MOVLW 0x80
MOVWF TRISA
MOVLW 0xF6          ; Настройка порта B.
MOVWF TRISB
BCF STATUS, RP1     ; Выбор банка 0.
BCF STATUS, RP0
CLRF PORTB

;Настройка приемопередатчика USART

BSF RCSTA, SPEN     ; Настройка приемника.
BCF RCSTA, RX9
BSF RCSTA, CREN
BCF STATUS, RP1     ; Выбор банка 1.
BSF STATUS, RP0
BCF TXSTA, TX9         ; Настройка передатчика.
BCF TXSTA, SYNC
BSF TXSTA, BRGH
MOVLW 0x16
MOVWF SPBRG
BCF STATUS, RP1     ; Выбор банка 0.
BCF STATUS, RP0
CLRW              ; Считывание собственного адреса.
ADDWF PORTB, 0      ; Прочитаем порт B.
ANDLW 0xF0          ; Нам не нужны младшие биты.
MOVWF 0x20          ; Сохраним адрес в 20h.
SWAPF 0x20, 1          ; Нам не нужна работа с младшими
битами.
CALL adrsim         ; Преобразуем его в символьный вид.

; прочитаем из EEPROM 30h - состояние реле.

BSF STATUS, RP0     ; Выбор банка 1.
BCF STATUS, RP1
MOVLW 0x00
MOVWF EEADR         ; Адрес считываемого регистра.
BSF EECON1, RD      ; Чтение.
```

```
MOVF EEDATA,W       ; w = EEDATA.
BCF STATUS, RP1     ; Выбор банка 0.
BCF STATUS, RP0
MOVWF 0x30
COMF 0x30,1         ; Будем хранить в EEPROM в инверсном виде.
CLRW
ADDWF 0x30,0        ; А в регистре 30h в прямом.
MOVWF PORTA         ; Перепишем 30h в порт.

; Начало работы, ожидание команды, отработка первой команды.

CLRW              ; Очищаем аккумулятор, ждем команды по USART.

start: BTFSS PIR1, RCIF   ; Ждем прихода первого символа
команды.
   GOTO start
   BCF STATUS, RP1 ; Выбор банка 0.
   BCF STATUS, RP0
   CALL cmnd     ;С приходом первого символа начинаем
обработку.
   GOTO start
NOP

; Обработка команды - проверка адреса, определение команды.

cmnd: BCF STATUS, Z
   MOVF RCREG, 0
   XORLW 0x52      ; Проверим, наш ли модуль R (52h).
   BTFSS STATUS, Z ; Если нет, вернемся
   RETURN

in1:  BTFSS PIR1, RCIF   ; Ждем прихода первого символа
адреса.
   GOTO in1     ; Если совпадает, продолжим.
   MOVF RCREG, 0
   BCF STATUS, Z
   XORWF 0x21, 0      ; Первый символ адреса.
   BTFSS STATUS, Z
   RETURN

in2:  BTFSS PIR1, RCIF   ; Ждем прихода второго символа
адреса.
   GOTO in2     ; Если совпадает, продолжим.
```

```
   MOVF RCREG, 0
   BCF STATUS, Z
   XORWF 0x22, 0       ; Второй символ адреса.
   BTFSS STATUS, Z
   RETURN

in3:  BTFSS PIR1, RCIF   ; Ждем прихода символа.
   GOTO in3  ; Если за адресом следует символ команды,
продолжим.
   MOVF RCREG, 0
   BCF STATUS, Z
   XORLW 0x24       ; Выполнение $(24h).
   BTFSC STATUS, Z
   CALL swtch       ; Вызываем подпрограмму выполнения.
   RETURN

;Подпрограмма перевода адреса в символы, их храним в 21h,
22h.

adrsim:  CLRW      ; Если адрес 1, запишем символы 01 (30h и
31h).
      ADDLW 0x30
      MOVWF 0x21
      CLRW
      ADDLW 0x31
      MOVWF 0x22
      MOVF 0x20, 0
      BCF STATUS, Z
      XORLW 0x1
      BTFSC STATUS, Z
      RETURN

      CLRW      ; Если адрес 2, запишем символы 02 (30h и
32h).
      ADDLW 0x30
      MOVWF 0x21
      CLRW
      ADDLW 0x32
      MOVWF 0x22
      MOVF 0x20, 0
      BCF STATUS, Z
      XORLW 0x2
      BTFSC STATUS, Z
```

```
RETURN

CLRW      ; Если адрес 3, запишем символы 03.
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x33
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x3
BTFSC STATUS, Z
RETURN

CLRW      ; Если адрес 4, запишем символы 04.
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x34
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x4
BTFSS STATUS, Z
RETURN

CLRW      ; Если адрес 5, запишем символы 05.
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x35
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x5
BTFSS STATUS, Z
RETURN

CLRW      ; Если адрес 6, запишем символы 06.
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x36
```

```
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x6
BTFSS STATUS, Z
RETURN

CLRW      ; Если адрес 7, запишем символы 07.
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x37
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x7
BTFSS STATUS, Z
RETURN

CLRW      ; Если адрес 8, запишем символы 08.
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x38
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x8
BTFSS STATUS, Z
RETURN

CLRW      ; Если адрес 9, запишем символы 09.
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x39
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x9
BTFSS STATUS, Z
RETURN
```

```
CLRW      ; Если адрес 10 (A), запишем символы 10.
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x30
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xA
BTFSS STATUS, Z
RETURN

CLRW      ; Если адрес 11, запишем символы 11.
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x31
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xB
BTFSS STATUS, Z
RETURN

CLRW      ; Если адрес 12, запишем символы 12.
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x32
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xC
BTFSS STATUS, Z
RETURN

CLRW      ; Если адрес 13, запишем символы 13.
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x33
MOVWF 0x22
MOVF 0x20, 0
```

```
        BCF STATUS, Z
        XORLW 0xD
        BTFSS STATUS, Z
        RETURN

        CLRW       ; Если адрес 14, запишем символы 14.
        ADDLW 0x31
        MOVWF 0x21
        CLRW
        ADDLW 0x34
        MOVWF 0x22
        MOVF 0x20, 0
        BCF STATUS, Z
        XORLW 0xE
        BTFSS STATUS, Z
        RETURN

        CLRW       ; Если адрес 15, запишем символы 15.
        ADDLW 0x31
        MOVWF 0x21
        CLRW
        ADDLW 0x35
        MOVWF 0x22
        MOVF 0x20, 0
        BCF STATUS, Z
        XORLW 0xF
        BTFSS STATUS, Z
        RETURN

; Подпрограмма включение реле по номеру.

cmdset:     CLRW          ; Запишем в 30h (установить бит).
        ADDWF 0x23, 0
        BCF STATUS, Z
        XORLW 0x0    ; Реле 0.
        BTFSC STATUS, Z
        BSF 0x30, 0
        CLRW
        ADDWF 0x23, 0
        BCF STATUS, Z
        XORLW 0x1    ; Реле 1.
        BTFSC STATUS, Z
        BSF 0x30, 1
```

```
      CLRW
      ADDWF 0x23, 0
      BCF STATUS, Z
      XORLW 0x2    ; Реле 2.
      BTFSC STATUS, Z
      BSF 0x30, 2
      CLRW
      ADDWF 0x23, 0
      BCF STATUS, Z
      XORLW 0x3    ; Реле 3.
      BTFSC STATUS, Z
      BSF 0x30, 3
      CLRW
      ADDWF 0x23, 0
      BCF STATUS, Z
      XORLW 0x4    ; Реле 4.
      BTFSC STATUS, Z
      BSF 0x30, 4
      CLRW
      ADDWF 0x23, 0
      BCF STATUS, Z
      XORLW 0x5    ; Реле 5.
      BTFSC STATUS, Z
      BSF 0x30, 5
      CLRW
      ADDWF 0x23, 0
      BCF STATUS, Z
      XORLW 0x6    ; Реле 6.
      BTFSC STATUS, Z
      BSF 0x30, 6
      CLRW
      ADDWF 0x23, 0
      BCF STATUS, Z
      XORLW 0x7    ; Реле 7.
      BTFSC STATUS, Z
      BSF 0x30, 7
      RETURN

; Подпрограмма выключения реле по номеру.

cmdreset: CLRW          ; Запишем в 30h (сбросить бит).
      ADDWF 0x23,0
      BCF STATUS, Z
```

```
XORLW 0x0    ; Реле 0.
BTFSC STATUS, Z
BCF 0x30, 0
CLRW
ADDWF 0x23,0
BCF STATUS, Z
XORLW 0x1    ; Реле 1.
BTFSC STATUS, Z
BCF 0x30, 1
CLRW
ADDWF 0x23,0
BCF STATUS, Z
XORLW 0x2    ; Реле 2.
BTFSC STATUS, Z
BCF 0x30, 2
CLRW
ADDWF 0x23,0
BCF STATUS, Z
XORLW 0x3    ; Реле 3.
BTFSC STATUS, Z
BCF 0x30, 3
CLRW
ADDWF 0x23,0
BCF STATUS, Z
XORLW 0x4    ; Реле 4.
BTFSC STATUS, Z
BCF 0x30, 4
CLRW
ADDWF 0x23,0
BCF STATUS, Z
XORLW 0x5    ; Реле 5.
BTFSC STATUS, Z
BCF 0x30, 5
CLRW
ADDWF 0x23,0
BCF STATUS, Z
XORLW 0x6    ; Реле 6.
BTFSC STATUS, Z
BCF 0x30, 6
CLRW
ADDWF 0x23,0
BCF STATUS, Z
XORLW 0x7    ; Реле 7.
```

```
    BTFSC STATUS, Z
    BCF 0x30, 7
    RETURN

; Подпрограмма определения команды N (включить),
F (выключить)
; или S (состояние).

swtch: CLRW
in4:  BTFSS PIR1, RCIF   ; Ждем прихода символа.
   GOTO in4
   MOVF RCREG, 0    ; Прочитаем номер реле.
   MOVWF 0x23       ; Сохраним номер реле в 23h.
   MOVLW 0x30       ; Запишем 30h в аккумулятор.
   SUBWF 0x23, 0       ; Переведем символ в номер.
   MOVWF 0x23       ; Сохраним номер реле в 23h.

in5:  BTFSS PIR1, RCIF   ; Ждем прихода символа.
   GOTO in5
   MOVF RCREG, 0    ; Прочитаем команду N-включить F-выключить.
   MOVWF 0x24       ; Сохраним команду в 24h.
   BCF STATUS, Z
   XORLW 0x4E       ; Включение N (4Eh).
   BTFSC STATUS, Z ; Если не включить, то пропустить.
   CALL cmdset
   MOVF 0x24, 0     ; Перепишем из 24h в аккумулятор.
   BCF STATUS, Z
   XORLW 0x46       ; Выключение F (46h).
   BTFSC STATUS, Z ; Если не выключить, то пропустить.
   CALL cmdreset
   MOVF 0x24, 0     ; Перепишем из 24h в аккумулятор.
   BCF STATUS, Z
   XORLW 0x53       ; S (53h) запрос статуса.
   BTFSC STATUS, Z ; Если не запрос статуса, то пропустить.
   CALL stat

; Сохраним состояние всех реле в энергонезависимой памяти.

   CLRW          ; Запишем 30h - состояние реле в EEPROM.
   BSF STATUS, RP0 ; Выбор банка 1.
   BCF STATUS, RP1
   MOVLW 0x00       ; Запишем 0h в аккумулятор.
   MOVWF EEADR      ; Запишем адрес 0h в регистр адреса.
```

```
    BCF STATUS, RP1 ; Выбор банка 0.
    BCF STATUS, RP0
    CLRW
    ADDWF 0x30,0    ; Запишем содержимое 30h в аккумулятор.
    COMF 0x30,0     ; Инвертируем перед сохранением.
    BSF STATUS, RP0 ; Выбор банка 1.
    BCF STATUS, RP1
    MOVWF EEDATA
    BSF EECON1, WREN   ; Разрешить запись.
    BCF INTCON, GIE ; Запретить прерывания.
    MOVLW 0x55
    MOVWF EECON2    ; Записать 55h.
    MOVLW 0xAA
    MOVWF EECON2    ; Записать AAh.
    BSF EECON1, WR     ; Установить флаг для начала запись.
    BSF INTCON, GIE ; Разрешить прерывания.
    BCF EECON1, WREN   ; Запретить запись.
    BCF STATUS, RP1 ; Выбор банка 0.
    BCF STATUS, RP0

chkwr: BTFSS PIR1, EEIF   ; Проверка завершения записи.
    GOTO chkwr
    CLRW
    ADDWF 0x30, 0
    MOVWF PORTA     ; Перепишем 30h в порт.
    BCF PIR1, EEIF  ; Сбросим флаг.
    RETURN

; Подпрограмма передачи статуса реле.

stat:  BCF STATUS, RP1 ; Выбор банка 0.
    BCF STATUS, RP0
    CLRW          ; Проверим реле 0.
    ADDWF 0x23, 0
    BCF STATUS, Z
    XORLW 0x0     ; Реле 0.
    BTFSC STATUS, Z
    CALL rel0
    CLRW          ; Проверим реле 1.
    ADDWF 0x23, 0
    BCF STATUS, Z
    XORLW 0x1     ; Реле 1.
    BTFSC STATUS, Z
```

```
    CALL rel1
    CLRW          ; Проверим реле 2.
    ADDWF 0x23, 0
    BCF STATUS, Z
    XORLW 0x2     ; Реле 2.
    BTFSC STATUS, Z
    CALL rel2
    CLRW          ; Проверим реле 3.
    ADDWF 0x23, 0
    BCF STATUS, Z
    XORLW 0x3     ; Реле 3.
    BTFSC STATUS, Z
    CALL rel3
    CLRW          ; Проверим реле 4.
    ADDWF 0x23, 0
    BCF STATUS, Z
    XORLW 0x4     ; Реле 4.
    BTFSC STATUS, Z
    CALL rel4
    CLRW          ; Проверим реле 5.
    ADDWF 0x23, 0
    BCF STATUS, Z
    XORLW 0x5     ; Реле 5.
    BTFSC STATUS, Z
    CALL rel5
    CLRW          ; Проверим реле 6.
    ADDWF 0x23, 0
    BCF STATUS, Z
    XORLW 0x6     ; Реле 6.
    BTFSC STATUS, Z
    CALL rel6
    CLRW          ; Проверим реле 7.
    ADDWF 0x23, 0
    BCF STATUS, Z
    XORLW 0x7     ; Реле 7.
    BTFSC STATUS, Z
    CALL rel7

; Теперь это все отправим в передатчик.

    BSF PORTB, 0    ; Переключим драйвер RS485 на передачу.
    BCF STATUS, RP1 ; Выбор банка 1.
    BSF STATUS, RP0
```

```
   BSF TXSTA, TXEN ; Разрешаем передачу.
   BCF STATUS, RP1 ; Выбор банка 0.
   BCF STATUS, RP0
   MOVLW 0x52
   MOVWF TXREG  ; Отправим символ модуля.
   BCF STATUS, RP1 ; Выбор банка 1.
   BSF STATUS, RP0

outdat0: BTFSS TXSTA, TXIF  ; Ждем отправки символа.
   GOTO outdat0
   BCF STATUS, RP1 ; Выбор банка 0.
   BCF STATUS, RP0
   CLRW
   ADDWF 0x21,0
   MOVWF TXREG     ; Отправим первый символ адреса.
   BCF STATUS, RP1 ; Выбор банка 1.
   BSF STATUS, RP0

outdat1: BTFSS TXSTA, TXIF  ; Ждем отправки символа.
   GOTO outdat1
   BCF STATUS, RP1 ; Выбор банка 0.
   BCF STATUS, RP0
   CLRW
   ADDWF 0x22,0
   MOVWF TXREG  ; Отправим второй символ адреса.
   BCF STATUS, RP1 ; Выбор банка 1.
   BSF STATUS, RP0

outdat2: BTFSS TXSTA, TXIF  ; Ждем отправки символа.
   GOTO outdat2
   BCF STATUS, RP1 ; Выбор банка 0.
   BCF STATUS, RP0
   MOVLW 0x23
   MOVWF TXREG  ; Отправим символ статуса.
   BCF STATUS, RP1 ; Выбор банка 1.
   BSF STATUS, RP0

outdat3: BTFSS TXSTA, TXIF  ; Ждем отправки символа.
   GOTO outdat3
   BCF STATUS, RP1 ; Выбор банка 0.
   BCF STATUS, RP0
   CLRW
   ADDWF 0x23,0
```

```
   ADDLW 0x30
   MOVWF TXREG      ; Отправим символ номера реле.
   BCF STATUS, RP1 ; Выбор банка 1.
   BSF STATUS, RP0

outdat4: BTFSS TXSTA, TXIF  ; Ждем отправки символа.
   GOTO outdat4
   BCF STATUS, RP1 ; Выбор банка 0.
   BCF STATUS, RP0
   CLRW
   ADDWF 0x25,0
   MOVWF TXREG      ; Отправим символ состояния реле.
   BCF STATUS, RP1 ; Выбор банка 1.
   BSF STATUS, RP0

outdat5: BTFSS TXSTA, RCIF  ; Ждем отправки символа.
   GOTO outdat5
   BCF TXSTA, TXEN ; Запрещаем передачу.
   BCF STATUS, RP1 ; Выбор банка 0.
   BCF STATUS, RP0
   BCF PORTB, 0     ; Переключим драйвер RS485 на прием.
   RETURN

;Подпрограмма обработки статуса реле.

rel0: BTFSC 0x30, 0
CALL sostn
   BTFSS 0x30, 0
   CALL sostf
   RETURN
rel1: BTFSC 0x30, 1
   CALL sostn
   BTFSS 0x30, 1
   CALL sostf
   RETURN
rel2: BTFSC 0x30, 2
   CALL sostn
   BTFSS 0x30, 2
   CALL sostf
   RETURN
rel3: BTFSC 0x30, 3
   CALL sostn
   BTFSS 0x30, 3
```

```
   CALL sostf
   RETURN
rel4: BTFSC 0x30, 4
   CALL sostn
   BTFSS 0x30, 4
   CALL sostf
   RETURN
rel5: BTFSC 0x30, 5
   CALL sostn
   BTFSS 0x30, 5
   CALL sostf
   RETURN
rel6: BTFSC 0x30, 6
   CALL sostn
   BTFSS 0x30, 6
   CALL sostf
   RETURN
rel7: BTFSC 0x30, 7
   CALL sostn
   BTFSS 0x30, 7
   CALL sostf
   RETURN

; Подпрограммы образования символов состояний.

sostn: MOVLW 0x4E   ; Состояние - включено.
   MOVWF 0x25
   RETURN
sostf: MOVLW 0x46   ; Состояние - выключено.
   MOVWF 0x25
   RETURN
END
```

Программа релейного модуля на языке С

Файл заголовка

```
#define MODULNAMESIM "R"
#define CMDSIM "$"

#define bitset(var,bitno) ((var) |= 1 << (bitno))
#define bitclr(var,bitno) ((var) &= ~(1 << (bitno)))
```

```
void putch(unsigned char);
unsigned char getch(void);
int init_comms();
int sim_num_adr();
int cmd();
int rel_on(int num);
int rel_off(int num);
int rel_stat(int num);
```

Основной файл

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "reley_c.h"

unsigned char input;          // Для считывания приемного
регистра.
unsigned char MOD_SIM1;       // Первый символ адреса модуля.
unsigned char MOD_SIM2;       // Второй символ адреса модуля.
unsigned char REL_SIM;           // Символ реле.
unsigned char command_reciev [6]; // Массив для полученной
команды.

int MOD_ADDR;              // Заданный адрес модуля, как число.
int sim_end_num = 0;          // Полный символьный номер
модуля.
int MOD_NUM;            // Полученный адрес модуля, как число.
int REL_NUM;         // Номер реле.
unsigned char RELSTAT = 0;  // Статус реле (позиционно):
1 - вкл, 0 - выкл.
int i;

// Получение байта.
unsigned char getch()
{
   while(!RCIF)    // Устанавливается, когда регистр не пуст.
      continue;
   return RCREG;
}

                // Вывод одного байта.
void putch(unsigned char byte)
{
```

```
   while(!TXIF)        // Устанавливается, когда регистр пуст.
      continue;
   TXREG = byte;
}

// Преобразуем символьный адрес в число.
int sim_num_adr()
{
   sim_end_num = 0;
   MOD_SIM1 = command_reciev [1]; // Первый символ номера.
   MOD_SIM2 = command_reciev [2]; // Второй символ номера.
   MOD_SIM1 = MOD_SIM1 - 0x30;    // В виде числа.
   MOD_SIM2 = MOD_SIM2 - 0x30;
   sim_end_num = MOD_SIM1*0x0A + MOD_SIM2;
   return sim_end_num;
}

// Получение и выполнение команды.
int cmd()
{
   REL_SIM = command_reciev [4];
   REL_NUM = REL_SIM - 0x30;
   switch (command_reciev [5])
{
      case "N": rel_on(REL_NUM);
      break;
      case "F": rel_off(REL_NUM);
      break;
      case "S": rel_stat(REL_NUM);
      break;
   }
}

// Выполнение команды включения заданного реле.
int rel_on(int num)
{
   bitset (RELSTAT, REL_NUM);
   PORTA = RELSTAT;
   EEADR =0x0;         // Запишем состояние реле в EEPROM.
   EEDATA = ~RELSTAT;
   WREN = 1;
   GIE = 0;
   EECON2 = 0x55;
```

```
    EECON2 = 0xAA;
    WR =1;
    while (WR);
    GIE = 1;
    WREN = 0;
}
//Выполнение команды выключения заданного реле.
int rel_off(int num)
{
    bitclr (RELSTAT, REL_NUM);
    PORTA = RELSTAT;
    EEADR =0x0;          // Запишем состояние реле в EEPROM.
    EEDATA = ~RELSTAT;
    WREN = 1;
    GIE = 0;
    EECON2 = 0x55;
    EECON2 = 0xAA;
    WR =1;
    while (WR);
    GIE = 1;
    WREN = 0;
}
// Выполнение команды передачи состояния заданного реле.
int rel_stat(int num)
{
    command_reciev[0] = "R";
    command_reciev[1] = MOD_SIM1+0x30;
    command_reciev[2] = MOD_SIM2+0x30;
    command_reciev[3] = "#";
    command_reciev[4] = REL_SIM;
    if ((RELSTAT>>REL_NUM)&0x01) command_reciev[5] = "N";
    if (!((RELSTAT>>REL_NUM)&0x01)) command_reciev[5] = "F";
    CREN =0;         // Запрещаем прием.
    RB0 = 1;         //Переключим драйвер RS485 на передачу.
    TXEN = 1;        // Разрешаем передачу.
    for (i=0; i<6; ++i)   putch(command_reciev[i]);
    for (i=0;i<1000;i++);     // Задержка для вывода   .
    for (i=0; i<6; ++i) command_reciev [i] = " ";
    RB0 = 0;         // Выключаем драйвер RS485 на передачу.
    TXEN = 0;        // Запрещаем передачу.
    CREN =1;         // Разрешаем прием.
}
```

```
int init_comms()           // Инициализация модуля.
{
PORTA = 0x0;        // Настройка портов A и B.
CMCON = 0x7;
TRISA = 0x0;
TRISB = 0xFE;
RCSTA = 0b10010000;        // Настройка приемника.
TXSTA = 0b00000110;        // Настройка передатчика.
SPBRG = 0x68;          // Настройка режима приема-передачи.
RB0 = 0;         // Выключаем драйвер RS485 на передачу.

          // Прочитаем состояние реле из EEPROM.
EEADR = 0x0;
RD = 1;
RELSTAT = ~EEDATA;
PORTA = RELSTAT;
}

void main(void)
{
init_comms();          // Инициализация модуля.
   for (i=0; i<6; ++i) command_reciev [i] = " ";
   command_reciev [0] = "R";
             //Прочитаем и преобразуем номер модуля.
   MOD_ADDR = PORTB;      // Номер модуля в старших битах.
   MOD_ADDR=MOD_ADDR>>4;     // Сдвинем на четыре бита.
             // Начинаем работать.
start: CREN =1;
   input = getch();
   switch (input)
   {
      case "R":    // Если обращение к релейному модулю.
      for (i=1; i<6; ++i)   // Запишем команду в массив.
         {
         input = getch();
         command_reciev [i] = input;
         }

      MOD_NUM = sim_num_adr();     // Чтение из сети.

      if (MOD_NUM != MOD_ADDR) break;    // Если не наш адрес.
      else
         if (command_reciev [3] = "$") cmd(); // Если команда.
```

```
        default: goto start;
   }
   goto start;
}
```

HEX-файл для загрузки в программатор

```
:1000000830100308A00042820308400383000D201D
:100010008301392B04068001840A0406031D0A2883
:020020000034AA
:1004E20083018C1E712A1A0808008301B700831247
:1004F20003130C1E782A370899000800F401F5014D
:100502000310F30CF20C031C8D2A7008F407710817
:10051200318710AF5070310F00DF10D7208730448
:1005220003190034812A8301850107309F00831655
:100532008501FE30860090308312980006308316C3
:100542009800683099008312061083169B011C14D0
:10055201A098312A200850008008301AD01AE01D1
:10056200 3008A0003108A100D030A007A1070A304E
:10057200F200F3012008F000F1017F222108740744
:10058200AD0075080318750AAE00F1002D08F000E1
:1005920008000130F000831203132908F100F10A68
:1005A200D42A0310F00DF10BD22A7008A2042208FB
:1005B200850083169B01831222098316 9A001C155B
:1005C2008B1355309D00AA309D009C149C18E72A7D
:1005D2008B171C1183120800130F00083120313E1
:1005E2002908F100F10AF72A0310F00DF10BF52AA0
:1005F2007009A205220885008316 9B0183122209 35
:1006020083169A001C158B1355309D00AA309D004D
:100612009C149C180A2B8B171C11831208008 3014F
:100622003308A300D030F000FF30F1002308700738
:10063200A90071080318710AAA002E2B2908B50017
:100642002A08B600CA2A2908B5002A08B600ED2AE7
:100652002908B5002A08B6008C2B3408463A03193B
:10066200242B083A03191F2B1D3A031D0800292BBE
:100672009422AB01AC01462B2B082F3E840083133E
:100682020308000AB0A0319AC0A2C08803AF00033
:1006920080307002063003192B02031C3D2B5230AE
:1006A200AF000608A500A6010430F000260DA60C36
:1006B200A50CF00B572B852BAB01AB0AAC012C0818
:1006C200803AF000803070020630031 92B020318C2
:1006D200762B7122A4002B082F3E8400831324085A
:1006E2008000AB0A0319AC0A602BAE227008A70087
```

```
:1006F2007108A8002606031D802B25082706031D66
:10070200852B2430B200102318167122A400523A0D
:1007120003195D2B852B52308301AF002008303E38
:10072200B0002108303EB1002330B2002308B300EC
:100732002208F0002908F100F10AA12B0310F00CA5
:10074200F10B9F2B7008F000701CA92B4E30B400E7
:100752002208F0002908F100F10AB12B0310F00C75
:10076200F10BAF2B7008F0007018B92B4630B400B3
:1007720018120614831698168312AB01AC012C08CA
:10078200803AF00080307002063003192B02031801
:10079200D42B2B082F3E8400831300087622AB0A49
:1007A2000319AC0AC02BAB01AC012C08803AF00053
:1007B20083307002E83003192B020318E42BAB0AD2
:1007C2000319AC0AD62BAB01AC012C08803AF0001D
:1007D20080307002063003192B020318FA2B2B0803
:1007E2002F3E8400831320308000AB0A0319AC0A29
:0E07F200E62B061083169812831218160800C4
:00000001FF
```

С теми, кому интересно провести эксперименты, не имея опыта работы с микроконтроллерами и программным обеспечением или желая более детально ознакомиться с модулем, мы продолжим разговор о том, как это все устроено.

Требования к модулю:

- модуль должен иметь реле, контакты которого позволят подключить нагрузку с параметрами: ~220В, 10А (мощности нагрузки ~ 2 кВт);
- модуль должен иметь сетевой интерфейс RS485 (системная сеть);
- по отношению к системной сети модуль должен быть пассивным, то есть его основным режимом работы будет прослушивание сети и выполнение команд, полученных по сети, если они адресованы модулю;
- адрес модуля задается установкой переключателя в соответствующее положение.

Поскольку контроллер PIC16F628A имеет встроенный USART, мы используем его для работы с интерфейсом RS485.

Адресный селектор можно организовать на микроконтроллере различным образом. Для начала используем наиболее очевидный способ – выводы порта, включенные на ввод

информации, соединим с контактами переключателя, который устанавливает их в состояние – «1» или «0». В дальнейшем мы обсудим другие способы задания адреса. Четыре бита дают возможность адресоваться к 16 устройствам.

Для подачи сетевых команд используем COM-порт компьютера с передачей байта (рис. 1.35).



Рис. 1.35. Биты данных RS232 из статьи на сайте http://measure.chat.ru

Каждый бит имеет длительность 4 мкс (при данной скорости).

Для обращения к релейным модулям используем дополнительный адресный префикс «R».

Практически, мы определились со всем необходимым, чтобы приступить к построению релейного модуля.

Команды, которые будет принимать наш модуль:

- включить – Rxx$xN;
- выключить – Rxx$xF.

Я добавлю команду передачи модулем состояния реле:

- передать статус – Rxx$xS.

При передаче статуса используем следующую символику:

- включено – Rxx#xN;
- выключено – Rxx#xF.

Здесь Rxx$xN означает: R – релейный модуль, xx – два символа адреса от 00 до 15, $ – символ команды (# – символ статуса), x – номер реле от 0 до 7, N – включить (F – выключить).

Уточним, как мы будем использовать ресурсы контроллера. Для ввода информации используем встроенный в контроллер USART.

Использование порта B:

- RB0 – управление передатчиком (микросхема MAX1483).
- RB1 – прием (приемник USART).
- RB2 – передача (передатчик USART).

Для ввода заданного адреса используем четыре старших бита порта B (RB7–RB4).

Использование порта A:

- для подключения реле используем биты порта A (RA0–RA7), что предполагает использование восьми реле;
- при необходимости увеличить количество реле можно использовать свободные (если они есть) биты порта B.

Мы готовы приступить к написанию программы.

Основные блоки программы:

- инициализация (конфигурация контроллера);
- ожидание и прием команды (ожидание активности в сети, чтение команды, проверка адреса, и, если адрес совпадает с адресом устройства, выполнение команды, возвращение к ожиданию команды);
- выполнение команды.

Распишем это поподробнее.

Инициализация.

В этом блоке мы должны задать конфигурацию контроллера, переход в режим низкого энергопотребления (SLEEP) и условия выхода из этого режима. В блоке инициализации мы прочитаем адрес, задаваемый адресным переключателем.

Ожидание и прием команды.

Этот блок программы является основным, поскольку в сети модуль играет пассивную роль, то есть ожидает прихода и выполняет команду, адресованную ему. При активации сети команда прочитывается. Если префикс совпадает с префиксом модуля, а адрес – с адресом устройства, команда выполняется, в противном случае модуль переходит в режим ожидания следующей сетевой команды. Адрес в слове команды сравнивается с ранее прочитанным (при инициализации) заданным адресом модуля.

Выполнение команды запроса статуса.

Этот блок программы отличается от предыдущего, начиная с момента получения символа S вместо N или F. Программа должна определить состояние соответствующего вывода порта, переключить USART на передачу и передать статус в формате Rxx#xN, если соответствующий вывод порта – в состоянии «1», или Rxx#xF, если он в состоянии «0». После этого программа переходит к ожиданию следующей команды.

Блок инициализации контроллера.

В первую очередь, определим, нужен ли нам режим «SLEEP», который переводит контроллер в состояние с уменьшенным энергопотреблением. Режим очень удобен и важен в тех случаях, когда контроллер используется в условиях батарейного питания. Переход в режим «SLEEP» продлевает срок службы батарей или время работы без подзарядки аккумулятора. В нашем случае нет необходимости в поддержании этого режима, поскольку модуль получает питание от блока питания.

Часть конфигурирования необходимо выполнить при программировании контроллера. Это относится к слову конфигурации по адресу 2007h. Здесь h после цифр означает HEX, hexadecimal (шестнадцатеричное число). В слове конфигурации устанавливаются (или не устанавливаются) биты защиты, выбирается режим работы тактового генератора и некоторые параметры, относящиеся к режиму питания контроллера.

Вот первый вариант слова конфигурации для тактовой частоты 16–20 МГц:

- Бит 13 устанавливаем в «1» – выключаем защиту кода.
- Бит 8 устанавливаем в «1» – выключаем защиту EEPROM.
- Бит 7 устанавливаем в «0» – вывод RB4 работает как цифровой канал ввода-вывода.
- Бит 6 устанавливаем в «0» – запрещаем сброс по снижению напряжения питания.
- Бит 5 устанавливаем в «0» – вывод RB5 работает как цифровой канал ввода-вывода.

- Бит 4 устанавливаем в «0» – режим HS генератора, кварц к выводам RA6 и RA7.
- Бит 3 устанавливаем в «0» – выключаем режим включения по таймеру.
- Бит 2 устанавливаем в «0» – выключаем режим работы сторожевого таймера.
- Бит 1 устанавливаем в «1» – режим HS генератора, кварц к выводам RA6 и RA7.
- Бит 0 устанавливаем в «0» – режим HS генератора, кварц к выводам RA6 и RA7.

Слово конфигурации будет выглядеть так – 3F0Ah.

Для работы внутреннего тактового генератора следует использовать кварцевый резонатор с параллельным резонансом на 16–20 МГц (рис. 1.36).

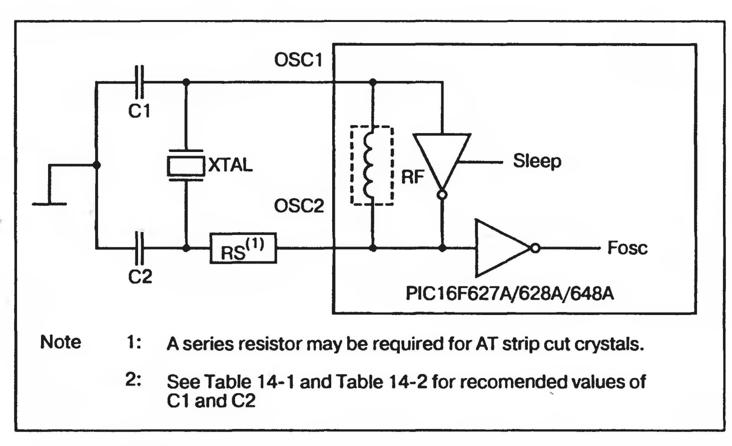


Рис. 1.36. Типовая схема включения кварцевого резонатора для PIC16F628A

Для частоты 16 МГц рекомендованные значения 10–22 пФ конденсаторов C1 и C2.

Давайте подумаем, а не использовать ли встроенный в контроллер генератор. Будем ли мы выполнять какие-либо операции, требующие стабильной тактовой частоты? На стабильность частоты внутреннего генератора, в первую очередь, влияет температура и напряжение питания. Будут ли они

значительно меняться? Если мы не планируем использовать модуль в уличном исполнении, температура меняется мало. Если же после входного напряжения 12 В мы поставим стабилизатор на 5 В, питающее напряжение будет изменяться еще меньше. В данный момент я больше склонен отказаться от кварцевого резонатора и использовать внутренний тактовый генератор.

В этом случае есть два варианта на выбор. Для использования режима ER (3,6 МГц) к выводу RA7 микросхемы подключается резистор (не более 40 кОм). Вывод RA6 может использоваться для ввода-вывода.

Второй вариант слова конфигурации:

- Биты 13–10 устанавливаем в «1» – выключаем защиту кода.
- Бит 8 устанавливаем в «1» – выключаем защиту EEPROM.
- Бит 7 устанавливаем в «0» – вывод RB4 работает как цифровой канал ввода–вывода.
- Бит 6 устанавливаем в «0» – запрещаем сброс по снижению напряжения питания.
- Бит 5 устанавливаем в «0» – вывод RB5 работает как цифровой канал ввода–вывода.
- Бит 4 устанавливаем в «1» – режим ER, резистор 40 кОм подключается к RA7.
- Бит 3 устанавливаем в «0» – выключаем режим включения по таймеру.
- Бит 2 устанавливаем в «0» – выключаем режим работы сторожевого таймера.
- Бит 1 устанавливаем в «1» – режим ER генератора, резистор подключается к RA7.
- Бит 0 устанавливаем в «0» – режим ER генератора, резистор подключается к RA7.

Слово конфигурации будет выглядеть так – 3F1Ah.

Еще более удобный вариант – использование внутреннего генератора без внешних элементов INTRC (4 МГц). Слово конфигурации в этом случае – 3F18h.

Здесь я хотел бы сделать замечание, или признание – как вам угодно. Я впервые буду использовать контроллер PIC16F628A и пользоваться средой программирования контроллера MPLAB. Может возникнуть закономерный вопрос – а чему я могу научить кого-то, если сам впервые это делаю? Если вас смущает данный аспект, можете обратиться к многочисленной литературе по программированию контроллеров, пропустив дальнейшее, что и решит проблему. Но мне кажется, что, имея большой опыт работы, основательно забываешь трудности, с которыми столкнулся в начале. А если сам только начинаешь работать, эти трудности налицо.

В блоке инициализации контроллера необходимо, в первую очередь, установить конфигурацию портов A и B.

Нам потребуется установить биты порта A 0–6 на вывод, бит 7 – на ввод (к выводу RA7 порта подключается резистор в режиме ER) или все выводы порта A – на вывод для режима генератора INTRC.

Для этого в регистр TRISA необходимо записать «1» для входных выводов и «0» для выходных. Ниже приведен пример инициализации для порта A, взятый из руководства:

```
CLRF PORTA   ;Initialize PORTA by setting output data latches
     ;(инициализация установкой защелок данных)
MOVLW 0x07   ;Turn comparators off and enable pins for I/O
functions
MOVWF CMCON  ;(выключение компараторов для I/O функций)

BCF STATUS, RP1
BSF STATUS, RP0 ;Select Bank1 (выбор банка 1)
MOVLW 0x80      ;Value used to initialize data direction
MOVWF TRISA     ;Set RA<4:0> as outputs TRISA<5> always read
as '1'.
         ;TRISA<7:6> depend on oscillator mode
         ; (в данном случае все на вывод, 7 ввод)
```

Здесь 0x07 означает, что число 7 – шестнадцатеричное.

Инициализация порта B связана, прежде всего, с необходимостью использования USART. Необходимо установить RB0 на

вывод и записать в него «0» (при передаче записывается «1»). Затем:

- RB1 устанавливается на ввод, а RB2 – на вывод;
- RB3 установим на вывод (для управления микросхемой MAX1483);
- RB4–RB7 устанавливаем на ввод для организации адресного селектора.

```
CLRF PORTB      ;Initialize PORTB by setting output data
latches
         ;(инициализация установкой защелок данных)
MOVLW 0xF2      ;Value used to initialize data direction
MOVWF TRISB
```

Запишем необходимые данные в регистр управления приемника USART – RCSTA (адрес 18h):

- Бит 7 – «1» – модуль включен;
- Бит 6 – «0» – 8-битный прием;
- Бит 5 – не имеет значения;
- Бит 4 – «1» – прием разрешен;
- Бит 3 – не имеет значения;
- Бит 2 – только на чтение;
- Бит 1 – только на чтение;
- Бит 0 – только на чтение.

Операции записи побитовые:

```
BSF RCSTA, SPEN
BCF RCSTA, RX9
BSF RCSTA, CREN
```

Аналогично запишем данные в регистр передатчика USART – TXSTA (адрес 98h):

- Бит 7 – не имеет значения;
- Бит 6 – «0» – 8-битная передача;
- Бит 5 – «1» – разрешение передачи;
- Бит 4 – «0» – асинхронный режим работы;
- Бит 3 – не используется;

- Бит 2 – «1» – высокоскоростной режим;
- Бит 1 – только чтение;
- Бит 0 – проверку четности мы не используем.

```
BCF TXSTA, TX9
BCF TXSTA, SYNC
BSF TXSTA, BRGH
BSF TXSTA, TXEN (эта команда появится, когда мы начнем
передачу статуса).
```

Кроме того, необходимо задать скорость работы в регистре SPBRG. Для первого варианта тактового генератора (SYNC = 0, BRGH = 1, 16 МГц):

```
MOVLW 0x67
MOVWF SPBRG
```

Для второго варианта тактового генератора (SYNC = 0, BRGH = 1, 3,6 МГц):

```
MOVLW 0x16
MOVWF SPBRG
```

В результате блок инициализации (для второго варианта) будет выглядеть следующим образом:

```
list    p=16f628a
#include p16f628a.inc

;Инициализация модуля

BCF STATUS, RP1 ; Выбор банка 0
BCF STATUS, RP0
```

Если посмотреть раздел организации памяти контроллера, видно, что часть регистров, которые мы используем, находится в банке «0», а часть в банке «1». Если забыть переключиться, команды могут не выполняться, а среда программирования MPLAB напомнит, выводя в окне **Output** на странице **MPLAB SIM** напоминания при трансляции программы. Это были первые грабли, на которые я часто наступал.

```
CLRF PORTA          ; Настройка порта A.

MOVLW 0x07
```

```
MOVWF CMCON

BCF STATUS, RP1    ; Выбор банка 1.
BSF STATUS, RP0

MOVLW 0x80
MOVWF TRISA

MOVLW 0xF6         ; Настройка порта B.
MOVWF TRISB

BCF STATUS, RP1    ; Выбор банка 0.
BCF STATUS, RP0
CLRF PORTB

;Настройка приемопередатчика USART.

BSF RCSTA, SPEN    ; Настройка приемника.
BCF RCSTA, RX9
BSF RCSTA, CREN

BCF STATUS, RP1    ; Выбор банка 1.
BSF STATUS, RP0

BCF TXSTA, TX9     ; Настройка передатчика.
BCF TXSTA, SYNC
BSF TXSTA, BRGH

MOVLW 0x16
MOVWF SPBRG

BCF STATUS, RP1    ; Выбор банка 0.
BCF STATUS, RP0

CLRW               ; Считывание собственного адреса.
ADDWF PORTB, 0     ; Прочитаем порт B.
ANDLW 0xF0         ; Нам не нужны младшие биты.
MOVWF 0x20         ; Сохраним адрес в регистре 20h.
SWAPF 0x20, 1      ; Нам не нужна работа с младшими битами.
CALL adrsim        ; Преобразуем его в символьный вид.
```

Чтобы сравнить адрес, задаваемый переключателем, с адресом в слове команды, нужно преобразовать их к единому

виду. Здесь существует несколько возможностей. Я использую одну из них, вы можете поступить иначе.

```
; прочитаем из EEPROM в 30h - состояние реле.
```

Необязательно хранить текущее состояние реле в энергонезависимой памяти. Но это может быть полезно и, кроме того, дает возможность разобраться с механизмом записи в энергонезависимую память. Можно, например, изменить метод задания адреса. Использовать вместо переключателя в адресном селекторе программную установку адреса – запись адреса в энергонезависимую память контроллера. Так, в частности, устроены некоторые промышленные модули. В этом случае перед первым использованием модуля его переводят в режим установки адреса и программно устанавливают этот адрес. Механизм записи состояния реле и адреса одинаков.

```
BSF STATUS, RP0 ; Выбор банка 1.
BCF STATUS, RP1
MOVLW 0x00
MOVWF EEADR      ; Адрес считываемого регистра.
BSF EECON1, RD ; Чтение.
MOVF EEDATA,W   ; w = EEDATA.
BCF STATUS, RP1 ; Выбор банка 0.
BCF STATUS, RP0
MOVWF 0x30
COMF 0x30,1      ; Будем хранить в EEPROM в инверсном виде.
CLRW
ADDWF 0x30,0     ; А в регистре 30h в прямом.
MOVWF PORTA      ; Перепишем 30h в порт.
```

Сохранение состояния реле в инверсном виде связано с тем, что в EEPROM по умолчанию записаны единицы. При инициализации прочтение состояния приведет к включению всех реле. Чтобы избежать этого, будем инвертировать прочитанное.

В строке «CALL adrsim» мы вызываем подпрограмму, которая переводит адреса в символы:

Это опять-таки не обязательно делать в данном программном блоке. Но оставлять в блоке инициализации длинный «хвост» мне не захотелось. Перевод адреса в символьное

представление можно сделать различными способами, я выбрал самый очевидный.

Поскольку мы используем символьный обмен, я выпишу шестнадцатеричные коды некоторых символов ASCII:

«0» – 30h; «1» – 31h; «2» – 32h; «3» – 33h; «4» – 34h; «5» – 35h и т. д.

«R» – 52h; «$» – 24h; «#» – 23h; «N» – 4Eh; «F» – 46h; «S» – 53h

```
; Подпрограмма перевода адреса в символ (храним по адресам
21h, 22h).

adrsim:   CLRW   ; Если адрес 1 запишем символы "0" "1" (30h и
31h).
      ADDLW 0x30
      MOVWF 0x21
      CLRW
      ADDLW 0x31
      MOVWF 0x22
      MOVF 0x20, 0
      BCF STATUS, Z
      XORLW 0x1
      BTFSC STATUS, Z
      RETURN

      CLRW   ; Если адрес 2 запишем символы "0" "2" (30h и 32h).
      ADDLW 0x30
      MOVWF 0x21
      CLRW
      ADDLW 0x32
      MOVWF 0x22
      MOVF 0x20, 0
      BCF STATUS, Z
      XORLW 0x2
      BTFSC STATUS, Z
      RETURN

      CLRW   ; Если адрес 3 запишем символы "0" "3".
      ADDLW 0x30
      MOVWF 0x21
      CLRW
      ADDLW 0x33
```

```
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x3
BTFSC STATUS, Z
RETURN

CLRW  ; Если адрес 4 запишем символы "0" "4".
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x34
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x4
BTFSS STATUS, Z
RETURN

CLRW  ; Если адрес 5 запишем символы "0" "5".
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x35
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x5
BTFSS STATUS, Z
RETURN

CLRW  ; Если адрес 6 запишем символы "0" "6".
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x36
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x6
BTFSS STATUS, Z
RETURN
```

```
CLRW   ; Если адрес 7 запишем символы "0" "7".
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x37
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x7
BTFSS STATUS, Z
RETURN

CLRW   ; Если адрес 8 запишем символы "0" "8".
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x38
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x8
BTFSS STATUS, Z
RETURN

CLRW   ; Если адрес 9 запишем символы "0" "9".
ADDLW 0x30
MOVWF 0x21
CLRW
ADDLW 0x39
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0x9
BTFSS STATUS, Z
RETURN

CLRW   ; Если адрес 10 (Ah) запишем символы "1" "0".
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x30
MOVWF 0x22
MOVF 0x20, 0
```

```
BCF STATUS, Z
XORLW 0xA
BTFSS STATUS, Z
RETURN

CLRW   ; Если адрес 11 (Bh) запишем символы "1" "1".
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x31
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xB
BTFSS STATUS, Z
RETURN

CLRW   ; Если адрес 12 (Ch) запишем символы "1" "2".
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x32
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xC
BTFSS STATUS, Z
RETURN

CLRW   ; Если адрес 13 (Dh) запишем символы "1" "3".
ADDLW 0x31
MOVWF 0x21
CLRW
ADDLW 0x33
MOVWF 0x22
MOVF 0x20, 0
BCF STATUS, Z
XORLW 0xD
BTFSS STATUS, Z
RETURN

CLRW   ; Если адрес 14 (Eh) запишем символы "1" "4".
ADDLW 0x31
```

```
    MOVWF 0x21
    CLRW
    ADDLW 0x34
    MOVWF 0x22
    MOVF 0x20, 0
    BCF STATUS, Z
    XORLW 0xE
    BTFSS STATUS, Z
    RETURN

    CLRW   ; Если адрес 15 (Fh) запишем символы "1" "5".
    ADDLW 0x31
    MOVWF 0x21
    CLRW
    ADDLW 0x35
    MOVWF 0x22
    MOVF 0x20, 0
    BCF STATUS, Z
    XORLW 0xF
    BTFSS STATUS, Z
    RETURN
```

Программный блок ожидания активности в сети.

1. Проверим бит RCIF в регистре PIR1.
2. Если бит установлен, пропустим следующую команду.
3. Вызовем подпрограмму обработки команды.
4. Вернемся к началу.

Вот и вся программа:

```
start: BTFSS PIR1, RCIF   ; Ждем прихода первого символа
команды.
   GOTO start
   CALL cmnd    ; С приходом первого символа начинаем
обработку.
   GOTO start
```

Следующая трудность спрятана как раз в этом месте, хотя я умудрился наступить на эти грабли при обработке «in1». Разумная, как мне казалось, конструкция:

```
start: BTFSS PIR1, RCIF   ; Ждем прихода первого символа
команды.
   GOTO start
```

работать в режиме отладки MPLAB не хотела. Я все перепроверил многократно. Все было правильно, но анимация программы безнадежно застревала, я решил, что программа не работает. Понять, в чем здесь дело, помогло обращение к сайту Microchip, где на форуме этот вопрос уже обсуждался. Причина столь не очевидного поведения программы очевидна (когда знаешь). Отладчик в цикле опроса ждет прохождения нескольких тысяч тактов внутреннего генератора, работающего на частоте в 4–20 МГц до заполнения регистра, которое происходит с частотой примерно 10 кГц. Вдобавок, настройки отладчика по умолчанию делают анимацию хорошо воспринимаемой (одно или два перемещения в секунду), но дождаться нескольких тысяч таких перемещений лично у меня ума не хватило.

Обработку команды я опять оформил в виде подпрограммы.

```
; Обработка команды - проверка адреса, определение команды.

cmnd:    BCF STATUS, Z
      MOVF RCREG, 0
      XORLW 0x52      ; Проверим наш ли модуль R (52h).
      BTFSS STATUS, Z ; Если нет вернемся.
      RETURN

in1:     BTFSS PIR1, RCIF   ; Ждем прихода первого символа
адреса.
      GOTO in1     ; Если совпадает, продолжим.
      MOVF RCREG, 0
      BCF STATUS, Z
      XORWF 0x21, 0      ; Первый символ адреса в регистре 21h.
      BTFSS STATUS, Z
      RETURN

in2:     BTFSS PIR1, RCIF   ; Ждем прихода второго символа
адреса.
      GOTO in2     ; Если совпадает, продолжим.
      MOVF RCREG, 0
      BCF STATUS, Z
      XORWF 0x22, 0      ; Второй символ адреса в регистре 22h.
      BTFSS STATUS, Z
      RETURN
```

```
in3:       BTFSS PIR1, RCIF    ; Ждем прихода символа.
      GOTO in3     ;Если следом символ команды, продолжим.
      MOVF RCREG, 0
      BCF STATUS, Z
      XORLW 0x24         ; Символ команды "$" (24h).
      BTFSC STATUS, Z
      CALL swtch         ; Вызываем подпрограмму выполнения.
      RETURN

; Подпрограмма определения команды N (включить), F
(выключить)
; или S (состояние).

swtch:    CLRW
in4:       BTFSS PIR1, RCIF    ; Ждем прихода символа.
      GOTO in4

      MOVF RCREG, 0         ; Прочитаем номер реле.
      MOVWF 0x23         ; Сохраним номер реле в регистре 23h.
      MOVLW 0x30         ; Запишем регистр 30h в аккумулятор.
      SUBWF 0x23, 0         ; Переведем символ в номер.
      MOVWF 0x23         ; Сохраним номер реле в регистре 23h.

in5:       BTFSS PIR1, RCIF    ; Ждем прихода символа.
      GOTO in5
      MOVF RCREG, 0 ; Разберем N-включить, F-выключить, S-
статус.
      MOVWF 0x24         ; Сохраним команду в регистре 24h.
      BCF STATUS, Z
      XORLW 0x4E         ; Включение N (4Eh).
      BTFSC STATUS, Z ; Если нет, то пропустим.
      CALL cmdset
      MOVF 0x24, 0       ; Перепишем из 24h в аккумулятор.
      BCF STATUS, Z
      XORLW 0x46         ; Выключение F (46h).
      BTFSC STATUS, Z ; Если нет, то пропустим.
      CALL cmdreset
      MOVF 0x24, 0       ; Перепишем из 24h в аккумулятор.
      BCF STATUS, Z
      XORLW 0x53         ; S (53h) запрос статуса.
      BTFSC STATUS, Z ; Если нет, то пропустим.
      CALL stat
```

```
; Сохраним состояние всех реле в энергонезависимой памяти.
; Перепишем регистр 30h состояния реле в EEPROM.

      CLRW
      MOVLW 0x00       ; Запишем 0h в аккумулятор.
      MOVWF EEADR      ; Запишем адрес 0h в регистр адреса.
      BCF STATUS, RP1 ; Выбор банка 0.
      BCF STATUS, RP0
      CLRW
      ADDWF 0x30,0     ; Запишем содержимое 30h в аккумулятор.
      COMF 0x30,0      ; Инвертируем перед сохранением.
      BSF STATUS, RP0 ; Выбор банка 1.
      BCF STATUS, RP1
      MOVWF EEDATA
      BSF STATUS, RP0
      BCF STATUS, RP1 ; Выбор банка 1.
      BSF EECON1, WREN   ; Разрешить запись.
      BCF INTCON, GIE ; Запретить прерывания.
      MOVLW 0x55
      MOVWF EECON2     ; Записать 55h.
      MOVLW 0xAA
      MOVWF EECON2     ; Записать AAh.
      BSF EECON1, WR  ; Установить флаг для начала запись.
      BSF INTCON, GIE ; Разрешить прерывания.
      BCF EECON1, WREN   ; Запретить запись.

; Запись 55h и AAh относятся к обязательным при работе с
EEPROM.

      BCF STATUS, RP1 ; Выбор банка 0.
      BCF STATUS, RP0

chkwr:   BTFSS PIR1, EEIF   ; Проверка завершения записи.
      GOTO chkwr
      CLRW
      ADDWF 0x30, 0
      MOVWF PORTA      ; Перепишем 30h в порт.
      BCF PIR1, EEIF  ; Сбросим флаг.
      RETURN
```

В этом месте работы с отладчиком я вначале проверил выполнение двух команд подряд. Все работало. Затем я вписал запись состояния реле в EEPROM с проверкой окончания

записи. И, естественно, обнаружил, что вторая команда перестала выполняться. Но теперь я быстрее сообразил, что запись занимает время, в течение которого успевает пройти необработанная часть команды. Я не стал «городить огород», вставив между двумя нужными командами третью, ненужную. Хочу еще заметить, что изготовитель микросхем советует осуществить проверку записи в EEPROM, которая вставляется после проверки завершения записи. Я этого не сделал. Но после создания прототипа я могу переделать программу, удалив из слова команды символьное представление номера модуля и номера реле, могу сделать и программное задание адреса. Тогда и добавлю проверку правильности записи в EEPROM.

Три подпрограммы выполнения команд

```
; Подпрограмма включение реле по номеру.
; Запишем в регистр 30h (установить соответствующий номеру
реле бит).

cmdset:         CLRW
         ADDWF 0x23, 0
         BCF STATUS, Z
         XORLW 0x0    ; Реле 0.
         BTFSC STATUS, Z
         BSF 0x30, 0
         CLRW
         ADDWF 0x23, 0
         BCF STATUS, Z
         XORLW 0x1    ; Реле 1.
         BTFSC STATUS, Z
         BSF 0x30, 1
         CLRW
         ADDWF 0x23, 0
         BCF STATUS, Z
         XORLW 0x2    ; Реле 2.
         BTFSC STATUS, Z
         BSF 0x30, 2
         CLRW
         ADDWF 0x23, 0
         BCF STATUS, Z
         XORLW 0x3    ; Реле 3.
         BTFSC STATUS, Z
```

```
        BSF 0x30, 3
        CLRW
        ADDWF 0x23, 0
        BCF STATUS, Z
        XORLW 0x4    ; Реле 4.
        BTFSC STATUS, Z
        BSF 0x30, 4
        CLRW
        ADDWF 0x23, 0
        BCF STATUS, Z
        XORLW 0x5    ; Реле 5.
        BTFSC STATUS, Z
        BSF 0x30, 5
        CLRW
        ADDWF 0x23, 0
        BCF STATUS, Z
        XORLW 0x6    ; Реле 6.
        BTFSC STATUS, Z
        BSF 0x30, 6
        CLRW
        ADDWF 0x23, 0
        BCF STATUS, Z
        XORLW 0x7    ; Реле 7.
        BTFSC STATUS, Z
        BSF 0x30, 7
        RETURN

; Подпрограмма выключения реле по номеру.
; Запишем в регистр 30h (сбросить соответствующий номеру реле
бит).

cmdreset:   CLRW
        ADDWF 0x23,0
        BCF STATUS, Z
        XORLW 0x0    ; Реле 0.
        BTFSC STATUS, Z
        BCF 0x30, 0
        CLRW
        ADDWF 0x23,0
        BCF STATUS, Z
        XORLW 0x1    ; Реле 1.
        BTFSC STATUS, Z
        BCF 0x30, 1
```

```
        CLRW
        ADDWF 0x23,0
        BCF STATUS, Z
        XORLW 0x2    ; Реле 2.
        BTFSC STATUS, Z
        BCF 0x30, 2
        CLRW
        ADDWF 0x23,0
        BCF STATUS, Z
        XORLW 0x3    ; Реле 3.
        BTFSC STATUS, Z
        BCF 0x30, 3
        CLRW
        ADDWF 0x23,0
        BCF STATUS, Z
        XORLW 0x4    ; Реле 4.
        BTFSC STATUS, Z
        BCF 0x30, 4
        CLRW
        ADDWF 0x23,0
        BCF STATUS, Z
        XORLW 0x5    ; Реле 5.
        BTFSC STATUS, Z
        BCF 0x30, 5
        CLRW
        ADDWF 0x23,0
        BCF STATUS, Z
        XORLW 0x6    ; Реле 6.
        BTFSC STATUS, Z
        BCF 0x30, 6
        CLRW
        ADDWF 0x23,0
        BCF STATUS, Z
        XORLW 0x7    ; Реле 7.
        BTFSC STATUS, Z
        BCF 0x30, 7
        RETURN

; Подпрограмма передачи статуса реле

stat:   BCF STATUS, RP1    ; Выбор банка 0.
      BCF STATUS, RP0
      CLRW               ; Проверим реле 0.
```

```
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x0         ; Реле 0.
BTFSC STATUS, Z
CALL rel0
CLRW              ; Проверим реле 1.
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x1         ; Реле 1.
BTFSC STATUS, Z
CALL rel1
CLRW              ; Проверим реле 2.
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x2         ; Реле 2.
BTFSC STATUS, Z
CALL rel2
CLRW              ; Проверим реле 3.
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x3         ; Реле 3.
BTFSC STATUS, Z
CALL rel3
CLRW              ; Проверим реле 4.
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x4         ; Реле 4.
BTFSC STATUS, Z
CALL rel4
CLRW              ; Проверим реле 5.
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x5         ; Реле 5.
BTFSC STATUS, Z
CALL rel5
CLRW              ; Проверим реле 6.
ADDWF 0x23, 0
BCF STATUS, Z
XORLW 0x6         ; Реле 6.
BTFSC STATUS, Z
CALL rel6
CLRW              ; Проверим реле 7.
ADDWF 0x23, 0
```

```
      BCF STATUS, Z
      XORLW 0x7         ; Реле 7.
      BTFSC STATUS, Z
      CALL rel7

; теперь это все отправим в передатчик

      BSF PORTB, 0      ; Переключим драйвер RS485 на передачу.
      BCF STATUS, RP1     ; Выбор банка 1.
      BSF STATUS, RP0
      BSF TXSTA, TXEN     ; Разрешаем передачу.
      BCF STATUS, RP1     ; Выбор банка 0.
      BCF STATUS, RP0
      MOVLW 0x52
      MOVWF TXREG       ; Отправим символ модуля.
      BCF STATUS, RP1     ; Выбор банка 1.
      BSF STATUS, RP0

outdat0:  BTFSS TXSTA, TXIF      ; Ждем отправки символа.
      GOTO outdat0
      BCF STATUS, RP1      ; Выбор банка 0.
      BCF STATUS, RP0
      CLRW
      ADDWF 0x21,0
      MOVWF TXREG       ; Отправим первый символ адреса.
      BCF STATUS, RP1     ; Выбор банка 1.
      BSF STATUS, RP0

outdat1:  BTFSS TXSTA, TXIF      ; Ждем отправки символа.
      GOTO outdat1
      BCF STATUS, RP1     ; Выбор банка 0.
      BCF STATUS, RP0
      CLRW
      ADDWF 0x22,0
      MOVWF TXREG           ; Отправим второй символ адреса.
      BCF STATUS, RP1     ; Выбор банка 1.
      BSF STATUS, RP0

outdat2:  BTFSS TXSTA, TXIF      ; Ждем отправки символа.
      GOTO outdat2
      BCF STATUS, RP1     ; Выбор банка 0.
      BCF STATUS, RP0
      MOVLW 0x23
```

```
      MOVWF TXREG     ; Отправим символ статуса.
      BCF STATUS, RP1    ; Выбор банка 1.
      BSF STATUS, RP0

outdat3: BTFSS TXSTA, TXIF     ; Ждем отправки символа.
      GOTO outdat3
      BCF STATUS, RP1    ; Выбор банка 0.
      BCF STATUS, RP0
      CLRW
      ADDWF 0x23,0
      ADDLW 0x30
      MOVWF TXREG     ; Отправим символ номера реле.
      BCF STATUS, RP1    ; Выбор банка 1.
      BSF STATUS, RP0

outdat4: BTFSS TXSTA, TXIF     ; Ждем отправки символа.
      GOTO outdat4
      BCF STATUS, RP1    ; Выбор банка 0.
      BCF STATUS, RP0
      CLRW
      ADDWF 0x25,0
      MOVWF TXREG     ; Отправим символ статуса реле.
      BCF STATUS, RP1    ; Выбор банка 1.
      BSF STATUS, RP0

outdat5: BTFSS TXSTA, RCIF     ; Ждем отправки символа.
      GOTO outdat5
      BCF TXSTA, TXEN    ; Запрещаем передачу.
      BCF STATUS, RP1    ; Выбор банка 0.
      BCF STATUS, RP0
      BCF PORTB, 0    ; Переключим драйвер RS485 на прием.
      RETURN

; Подпрограмма обработки статуса реле.

rel0:    BTFSC 0x30, 0
         CALL sostn
         BTFSS 0x30, 0
         CALL sostf
         RETURN
rel1:    BTFSC 0x30, 1
         CALL sostn
         BTFSS 0x30, 1
         CALL sostf
```

```
        RETURN
rel2:   BTFSC 0x30, 2
        CALL sostn
        BTFSS 0x30, 2
        CALL sostf
        RETURN
rel3:   BTFSC 0x30, 3
        CALL sostn
        BTFSS 0x30, 3
        CALL sostf
        RETURN
rel4:   BTFSC 0x30, 4
        CALL sostn
        BTFSS 0x30, 4
        CALL sostf
        RETURN
rel5:   BTFSC 0x30, 5
        CALL sostn
        BTFSS 0x30, 5
        CALL sostf
        RETURN
rel6:   BTFSC 0x30, 6
        CALL sostn
        BTFSS 0x30, 6
        CALL sostf
        RETURN
rel7:   BTFSC 0x30, 7
        CALL sostn
        BTFSS 0x30, 7
        CALL sostf
        RETURN

; Подпрограммы образования символов состояний.

sostn:  MOVLW 0x4E      ; Состояние - включено.
        MOVWF 0x25
        RETURN
sostf:  MOVLW 0x46      ; Состояние - выключено.
        MOVWF 0x25
        RETURN
```

Если теперь соединить все представленные части программы, программирование контроллера завершится.

Вставки и примечания, которые я привел выше, появились позже, при наладке модуля, но я решил привести их «досрочно». Очень часто у меня не хватает терпения дочитать что-то до конца, мне не терпится начать работу. Я представил, что подобной чертой характера могу отличаться не только я, и привел все примечания на случай, если и вы уже начали работу за компьютером. Мне очень не хотелось, чтобы вы полностью разочаровались во всем преждевременно.

Мы написали программу для контроллера. Постараемся ее проверить и отладить в программе MPLAB.

Введение в работу с MPLAB

После загрузки программы появляется рабочее окно. Вид программы обычен для Windows и, думаю, не требует особых пояснений.

Мы создадим новый проект в основном меню **Project ➡ New** (Проект ➡ Новый). Задаем название relay проекту в папке Relay, которую я советую создать в основном разделе диска в папке MPLAB. Неоднократно я сталкивался с проблемой, которая не всегда очевидна. Многие программы, да это и удобно, предлагают хранить проект в папке Мои документы. Проблемы не возникает, если вы пользуетесь англоязычной версией Windows или русскоязычной версией программы. Но многие специализированные англоязычные программы начинают вытворять чудеса, если вы работаете в русскоязычной версии операционной системы. Впервые я столкнулся с этим, когда одна из сред программирования при компиляции программы стала выдавать ошибку в строке –1. Что она имела в виду под строкой с отрицательным номером, я не знаю. Но отыскать ошибку в правильно написанной программе оказалось не так просто. Ошибка крылась в том, что программа, предлагая сохранить проект в папке Мои документы, при компиляции эту папку распознать не могла.

После создания нового проекта появляется окно навигатор проекта. Теперь выберем микросхему контроллера в разделе основного меню **Configure ➡ Select Device** (Конфигурация ➡ Выбор устройства). Выбираем **PIC16F628A** (рис. 1.37).



Рис. 1.37. Рабочее окно программы MPLAB

Завершив выбор, следует создать файл основной программы. Выбираем **File ➡ New** (Файл ➡ Новый). Появляется окно редактора. Сохраним файл под именем relay.asm (**File ➡ Save As** (Файл ➡ Сохранить как...)). Добавим этот файл в проект. Для этого щелкнем правой кнопкой мыши раздел **Source Files** (Исходные файлы) в окне навигатора. В открывшемся меню выберем **Add Files** (Добавить файлы) и укажем свой файл.

Для начала перенесем в окно редактора небольшой фрагмент программы: инициализацию, ожидание команды и две подпрограммы: `CALL adrsim` и `CALL cmnd`. Последние подпрограммы – в усеченном виде. В данный момент будем использовать адрес модуля «01». Не забудьте поставить `END` в конце программы!

```
adrsim:  CLRW  ; Если адрес 1 запишем символы "0" "1" (30h и
31h).
      ADDLW 0x30
      MOVWF 0x21
      CLRW
      ADDLW 0x31
      MOVWF 0x22
      MOVF 0x20, 0
```

```
      BCF STATUS, Z
      XORLW 0x1
      BTFSC STATUS, Z
      RETURN

cmnd:     BCF STATUS, Z
      MOVF RCREG, 0
      XORLW 0x52       ; Проверим наш ли модуль R (52h).
      BTFSS STATUS, Z ; Если нет, вернемся.
      RETURN
in1:      BTFSS PIR1, RCIF   ; Ждем прихода первого символа
адреса.
      GOTO in1  ; Если совпадает, продолжим.
      MOVF RCREG, 0
      BCF STATUS, Z
      XORWF 0x21, 0        ; Первый символ адреса в регистре
21h.
      BTFSS STATUS, Z
      RETURN
in2:      BTFSS PIR1, RCIF   ; Ждем прихода второго символа
адреса.
      GOTO in2     ; Если совпадает, продолжим.
      MOVF RCREG, 0
      BCF STATUS, Z
      XORWF 0x22, 0        ; Второй символ адреса в регистре
22h.
      BTFSS STATUS, Z
      RETURN
```

Для отладки откроем окно наблюдения **View ➡ Watch** (Вид ➡ Наблюдение), в котором выберем регистры STATUS, WREG, PIR1, EEDATA, RCREG, 20h, 21h, 22h, 30h. Необходимые регистры открываются кнопкой с обозначением стрелки вниз, правее располагается названия регистра рядом с кнопкой **ADD SFR** (Добавить наблюдаемые). Ее следует щелкнуть после выбора регистра (рис. 1.38).

Регистры без имени (20h, 21h и т. д.) мы добавляем, просто вводя адрес в колонку **Address** (Адрес) на новой строке.

Добавим и EEPROM через выбор **View ➡ EEPROM** (Вид ➡ EEPROM). Установим в качестве текущего симулятора программный (**Debugger ➡ Select Tool ➡ MPLAB SIM** (Отладчик ➡ Выбор средства ➡ MPLAB SIM)). Создадим два файла input.txt

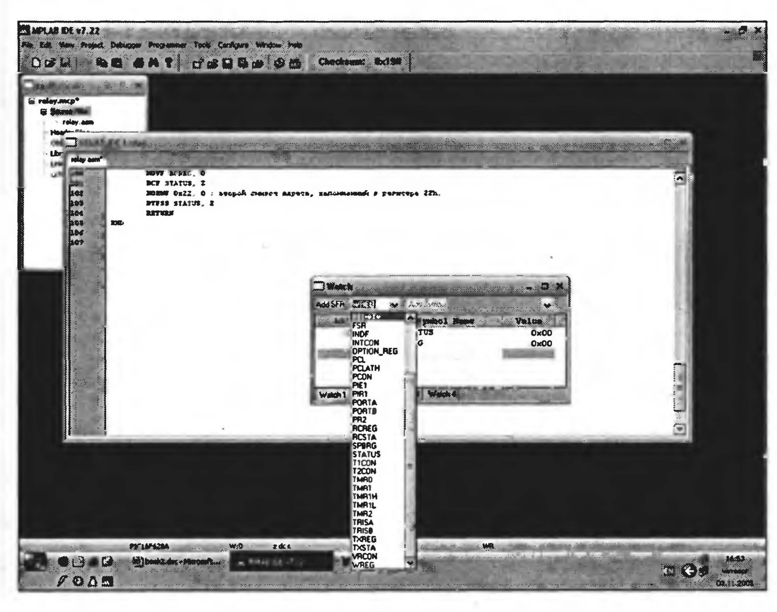


Рис. 1.38. Окно наблюдения **Watch**

и output.txt (**File** ➡ **New** (Файл ➡ Новый)). В файле input.txt, который открывается в окне редактора, запишем слово команды R01$ в виде строки в кавычках. Сохраним этот файл (**File** ➡ **Save** (Файл ➡ Сохранить)). Теперь сделаем установки отладчика (**Debugger** ➡ **Settings...** (Отладчик ➡ Установки...)): зададим частоту процессора 4 МГц, на вкладке **Uart1 IO** установим опцию **Enable Uart1 IO**, укажем входной файл input.txt (browse) и выходной output.txt. Подтвердим замену последнего файла и установим опцию **Rewind Input**. Изменим значение на вкладке **Animation/Realtime** на 1 мс. Щелкнем **Применить** и **OK**. Включим в окна **View** ➡ **Output** (Вид ➡ Вывод). Создадим новый сценарий, который позволит ввести адрес, имитируя переключатель. Для этого выберем в основном меню **Debugger** ➡ **Stimulus Controller** ➡ **New Scenario** (Отладчик ➡ Управление симуляцией ➡ Новый сценарий). Выберем вывод RB4 в окне **Pin/SFR** (Вывод/Наблюдаемые). В окне **Action** (Действие) выберем **Set High** (Установить в высокое состояние). Щелкнем кнопку со стрелкой вправо в колонке **Fire** (Запустить).

Сохраним сценарий под именем relay и свернем его (не закроем, а свернем!). Теперь откроем окно (**Configure** ➡ **Configuration Bits** (Конфигурация ➡ Биты конфигурации)), где установим биты, записываемые по адресу 2007h. Должно получиться слово 3F1Ah. Закроем это окно (рис. 1.39).

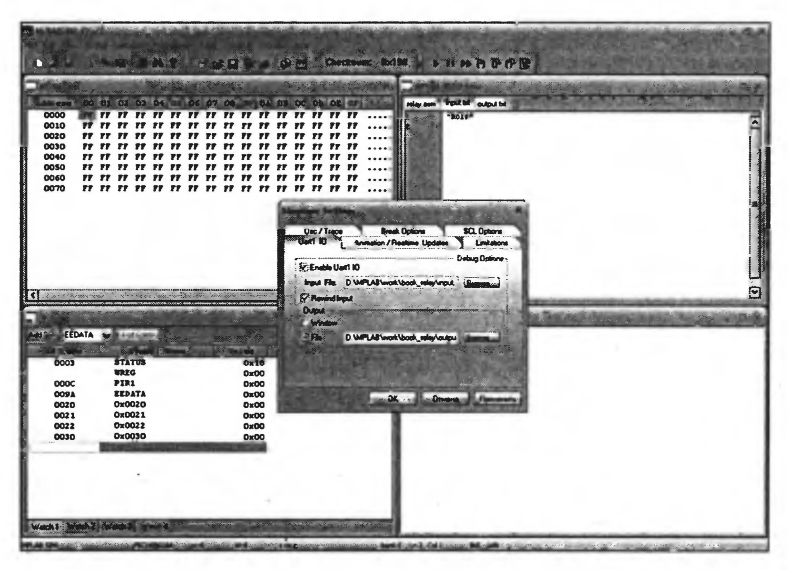


Рис. 1.39. Установки наблюдения и отладки

В завершение этих процедур упорядочим окна (**Windows** ➡ **Tile Horizontally** (Окна ➡ Расположить горизонтально)), сохраним все (**File** ➡ **Save All** (Файл ➡ Сохранить все)) и добавим в проект файл todo.txt, который предварительно создадим и поместим в раздел **Other Files** (Другие файлы) менеджера проекта. В файле todo.txt будем вести план работы.

Теперь откомпилируем проект **Project** ➡ **Build All** (Проект ➡ Компоновать все). Сохраним и вид проекта **File** ➡ **Save**

Workspace (Файл ➡ Сохранить рабочую область). Мы готовы к отладке программы.

После выхода из программы MPLAB и подтверждения сохранения вида проекта, новой загрузки программы и открытия проекта (**Project ➡ Open** (Проект ➡ Открыть)) мне приходится обнулять адрес 0h в EEPROM, щелкать **Fire** (Запустить) в **Scenario** (Сценарий), и вводить адреса 20h, 21h и т.д. в окне **Watch** (Наблюдение), которые, как пишет программа, **Not Found** (Не найдены). Я не уверен, что дернул за все веревочки, но... пока ничего лучшего не получилось.

Если вы правильно перенесли программу, то, щелкнув на инструментальной панели кнопку запуска программы, вы увидите анимацию, а в регистре RCREG (приемный регистр USART) появятся шестнадцатеричные коды символов из строки файла input.txt. Можно вписать в ячейку по адресу 0h EEPROM значение 1h и увидеть, как оно переписывается в регистр 30h.

Есть еще несколько полезных возможностей. Одна из них – проверить работу с определенного места. Для этого остановим анимацию (кнопкой паузы на инструментальной панели), установим курсор в нужной строке и щелкнем правой кнопкой мыши. Выберем в раскрывающемся меню **Set PC at Cursor** (Установить счетчик команд к курсору). Теперь, щелкая значок **Step Into** (Шаг внутрь) на инструментальной панели, мы можем отследить все изменения.

В окончательном виде я работаю в среде, которая показана на рис. 1.40.

Кроме написания программы на ассемблере среда MPLAB поддерживает написание программ на языке С. Существуют компиляторы разных производителей. Я использую демонстрационную версию кросскомпилятора **Hi-Tech.** Посмотрим, не будет ли проще написать программу на языке С?

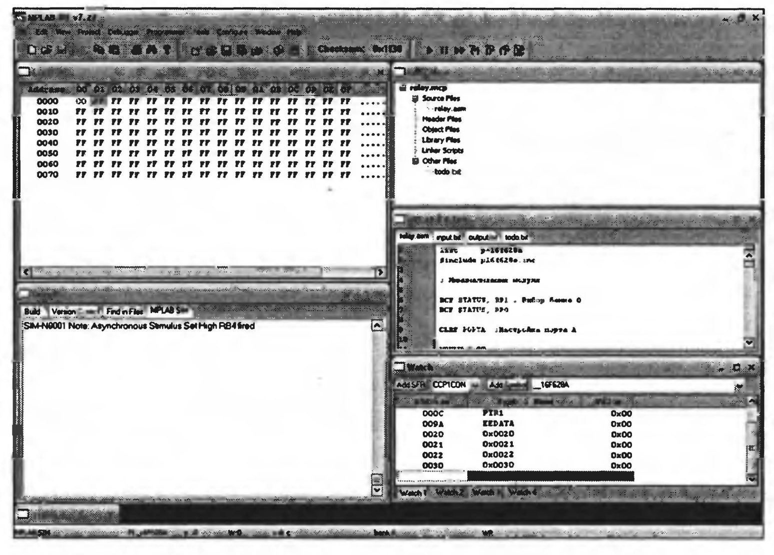


Рис. 1.40. Окончательный вид окна проекта

Релейный модуль, версия программы на языке С

Выбор языка программирования происходит при задании в **Project ➡ Select Language Toolsuite** (Проект ➡ Выбор языковых средств) – рис. 1.41.

Конечно, компилятор **Hi-Tech** должен быть установлен согласно инструкции.

Создадим файл заголовка и основной файл. Добавим в файлы заголовка нужный нам контроллер. Остальная часть работы мало, чем отличается от работы на ассемблере. В этой версии программы я не сохраняю состояние реле в EEPROM.

Файл заголовка

```
#define MODULNAMESIM "R"
#define CMDSIM "$"

#define bitset(var,bitno) ((var) |= 1 << (bitno))
```

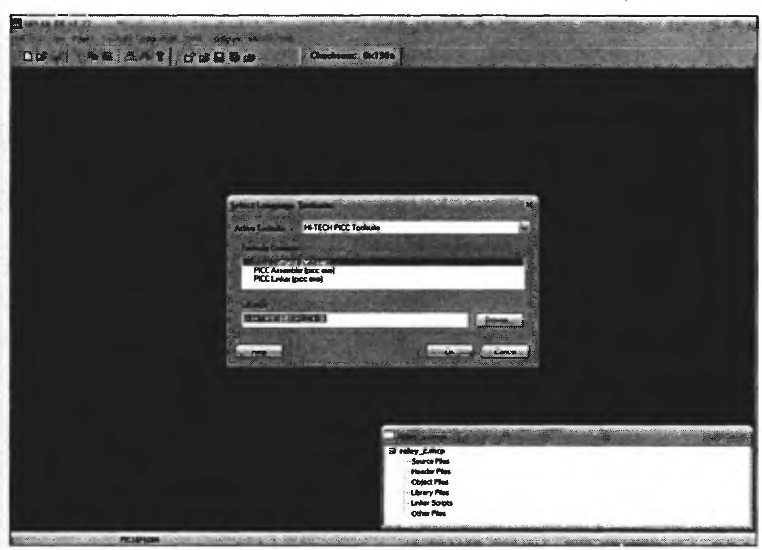


Рис. 1.41. Установка языка программирования С

```
#define bitclr(var,bitno) ((var) &= ~(1 << (bitno)))

void putch(unsigned char);
unsigned char getch(void);
int init_comms();
int sim_num_adr();
int cmd();
int rel_on(int num);
int rel_off(int num);
int rel_stat(int num);
```

Основной файл

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "reley_c.h"

unsigned char input;      // Считываем содержимое приемного
регистра.
unsigned char MOD_SIM1;   // Первый символ адреса модуля.
unsigned char MOD_SIM2;   // Второй символ адреса модуля.
unsigned char REL_SIM;    // Символ реле.
```

```
unsigned char COMMAND;    // Символ команды.
int MOD_ADDR;             // Заданный адрес модуля, как число.
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
int MOD_NUM;          // Полученный адрес модуля, как число.
int REL_NUM;          // Номер реле.
int RELSTAT = 0;        // Статус реле (позиционно): 1 - вкл, 0
- выкл.
              // Получение байта.

unsigned char getch()
{
   while(!RCIF)     // Устанавливается, когда регистр не пуст.
      continue;
   return RCREG;
}
              // Вывод одного байта.

void putch(unsigned char byte)
{
   PORTB = 1;       // Переключим драйвер RS485 на передачу.
   TXEN = 1;    // Разрешаем передачу.
   while(!TXIF)     // Устанавливается, когда регистр пуст.
      continue;
   TXREG = byte;
}

// Преобразуем символьный адрес в число.
int sim_num_adr()
{
   sim_end_num = 0;
   sim1_num = getch();        // Чтение первого символа номера.
   MOD_SIM1 = sim1_num;       // Сохраним первый символ.
   sim2_num = getch();        // Чтение второго символа номера.
   MOD_SIM2 = sim2_num;       // Сохраним второй символ.
   sim1_num = sim1_num - 0x30; // От первого символа к числу.
   sim2_num = sim2_num - 0x30; // От второго символа к числу.
   sim_end_num = sim1_num*0x0A + sim2_num;  //Объединим в одно
число.
   return sim_end_num;
}
```

```
// Получение и выполнение команды.
int cmd()
{
input = getch();
   REL_NUM = input;       // Номер реле в символьном виде.
   REL_SIM = REL_NUM;
   REL_NUM = REL_NUM - 0x30;   // Номер реле в числовом виде.
switch (COMMAND = getch())     // Прочитаем команду.
{
   case "N": rel_on(REL_NUM);  // Если команда включить.
   break;
   case "F": rel_off(REL_NUM); // Если команда выключить.
   break;
   case "S": rel_stat(REL_NUM);// Если команда передать
состояние.
   break;
   }
}

// Выполнение команды включения заданного реле.
int rel_on(int num)
{
   bitset (RELSTAT, REL_NUM);  // В RELSTAT побитовое
состояние реле.
PORTA = RELSTAT;      //Установив бит, переписываем в порт А.
}

//Выполнение команды выключения заданного реле.
int rel_off(int num)
{
   bitclr (RELSTAT, REL_NUM);
   PORTA = RELSTAT;      // Сбросив бит, переписываем в порт
А.
}

// Выполнение команды передачи состояния заданного реле.
int rel_stat(int num)
{
   putch("R");        // Отправляем символ R.
   putch(MOD_SIM1);      // Первый символ номера модуля.
   putch(MOD_SIM2);      // Второй символ номера модуля.
   putch("#");        // Символ статуса.
   putch(REL_SIM);    // Символ номера реле.
```

```
   if ((RELSTAT>>REL_NUM)&0x01) putch("N"); // Установлен ли
бит?
   if (!((RELSTAT>>REL_NUM)&0x01)) putch("F"); // Сброшен ли
бит?
      putch(0x0A);    // Только для вывода в файл!!!!!!
}

int init_comms()   // Инициализация модуля.
{
PORTA = 0x0;    //  Настройка портов А и В.
CMCON = 0x7;
TRISA = 0x80;
TRISB = 0xF6;
RCSTA = 0x90;      //  Настройка приемника.
TXSTA = 0x4;    //  Настройка передатчика.
SPBRG = 0x16;      //  Настройка режима приема-передачи.
INTCON=0;    //  Запретить прерывания.
PORTB = 0;      //  Выключим передатчик драйвера RS485.
}

void main(void)
{
init_comms();          // Инициализация модуля.

// Прочитаем и преобразуем номер модуля.
   MOD_ADDR = PORTB;     // Номер модуля в старших битах.
   MOD_ADDR=MOD_ADDR>>4;     // Сдвинем на четыре бита.

// Начинаем работать.
start: input = getch();
   while(input != MODULNAMESIM) input = getch();// Ждем, если
не нас.
   MOD_NUM = sim_num_adr();       //Чтение из сети (файла).
   if (MOD_NUM == MOD_ADDR)       // Если наш адрес модуля.
   {
      input = getch();
      if (input == CMDSIM) cmd();    // Если символ команды.
   }
   goto start;
}
```

Это не шедевр программирования на языке С, но работает. В качестве входного файла команд я использовал input.txt

(примечания ниже только для книжного варианта, в файле их делать не следует) такого вида:

```
"noperat" Проверим, не будет ли мешать посторонняя команда
"L03$3N"  Проверим, не отвечает ли модуль на обращение к
другим модулям
"R11$2N"  Проверим, не отвечает ли модуль на чужие адреса
"R03$2N"  Включим второе реле третьего релейного модуля
"R01$2S"  Проверим, не отвечает ли модуль на чужие адреса
"R03$2S"  Запросим состояние второго реле третьего релейного
модуля
"R15$1N"  Проверим, не отвечает ли модуль на чужие адреса
"R03$1N"  Включим первое реле третьего релейного модуля
"R03$1S"  Запросим состояние первого реле третьего релейного
модуля
"R03#1F"  Проверим, не отвечает ли модуль на передачу
состояния
"R03$1F"  Выключим первое реле третьего релейного модуля
"R03$1S"  Запросим состояние первого реле третьего релейного
модуля
"R03$2S"  Запросим состояние второго реле третьего релейного
модуля
```

Получаем выходной файл output.txt:

```
R03#2N
R03#1N
R03#1F
R03#2N
```

Теперь, пока вы опробуете работу в среде MPLAB, я, пожалуй, прерву работу над книгой. Поеду и куплю все необходимое для сборки конвертора RS232–RS485, программатора и создания прототипа. Затем соберу прототип на макетной плате, а когда закончу и проверю, поделюсь впечатлениями.

Первая сборка на макетной плате

Пришло время поделиться впечатлениями.

Подсчитав свои финансовые возможности, я отказался от некоторых запланированных покупок и решил упростить программатор, поскольку в настоящий момент собираюсь работать только с контроллером PIC16F628A. Я убрал из схемы

программатора внешнее питание, из схемы адаптера к программатору – все панельки, кроме 18-ножечной, и использовал батарейку «Крона» в качестве внутреннего источника питания (для высоковольтного режима программирования).

На макетной плате я тоже установил панельку под микросхему, с тем, чтобы проверить все схемы на одной макетной плате.

К схеме конвертера я добавил стабилизатор на 5 В. Кабель от разъема DB9 конвертера RS232–RS485 я распаял на плате конвертера, хотя вначале собирался использовать разъем. После исправления нескольких монтажных ошибок (и эти «грабли» имели место) я решил проверить программатор, поскольку не в полной мере был уверен, что правильно разобрался со схемой.

С программой MPLAB программатор, естественно, не работает. В Интернете есть схемы программаторов, работающих с MPLAB. Они не настолько сложны, чтобы их не использовать, но я решил, что лучше воспользоваться программой PonyProg2000, схему программатора к которой собрал. Приходится скачивать последнюю версию программы, поскольку в предыдущей нет контроллера PIC16F628A.

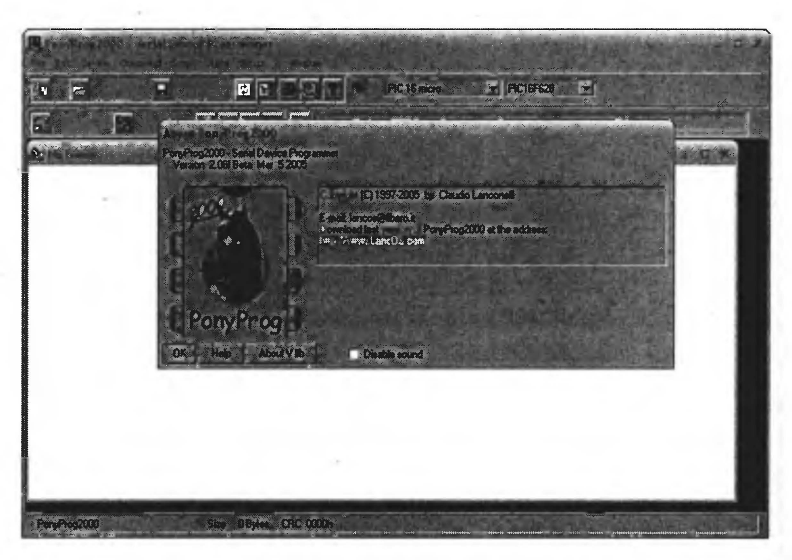


Рис. 1.42. Программа PonyProg2000

Первое, что я делаю после запуска программы – выбираю устройство в меню **Device ➡ Pic 16 micro** (Устройство ➡ Pic 16 micro) или в окошках на панели. После выбора устройства проверяю работу порта в меню **Setup (Interface Setup...** (Установки интерфейса), используя COM2-порт), задаю номер порта и **SI Prog I/O**. После щечка кнопки **Probe** (Проба) я получаю сообщение – **Test Ok**. Затем провожу калибровку (меню **Setup ➡ Calibration** (Установки ➡ Калибровка)) и опять получаю одобрение.

На этом везение заканчивается.

Вставляю микросхему в панельку, и первое, на что отваживаюсь – щелкаю кнопку чтения **Read Device** (Прочитать устройство) на инструментальной панели. Итог показан на рис. 1.43.

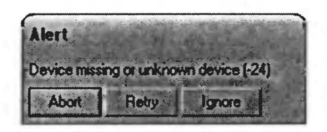


Рис. 1.43. Первое сообщение программы PonyProg2000

Программа не распознает микросхему. Обращаюсь на сайт, с которого «срисовал» программу. В разделе типовых вопросов и ответов нахожу похожую ситуацию и рекомендацию воспользоваться кнопкой **Ignore** (Игнорировать). Что и осуществляю.

Процесс проходит успешно, о чем сообщает программа, но в буфере сплошные *единицы*, а у меня сплошные сомнения – прочитал ли я хоть что-то? Я отключаю разъем от COM-порта, получаю сообщение об успешном чтении, но в буфере одни *нули*, а тест порта не проходит. Появляется надежда, что я что-то читаю.

Вторая волна сомнений касается вопроса: работает ли микросхема, если я не записал биты конфигурации, определяющие режим работы тактового генератора контроллера, ведь я еще ничего не записывал. Поэтому решаюсь записать биты конфигурации, выбрав режим работы тактового генератора **INTRC** (сэкономил, отказавшись от покупки кварца). В меню **Command ➡ Security and Configuration Bits...**

(Команда ➡ Биты конфигурации и защиты) устанавливаю флажки (рис. 1.44).

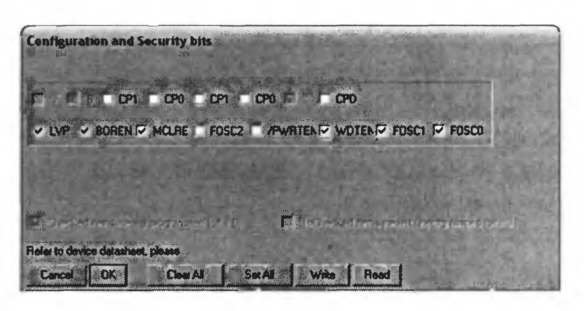


Рис. 1.44. Задание слова конфигурации микроконтроллера

Сознаюсь, что, не разглядев, как устанавливаются биты, я сделал все наоборот – поставил галочки там, где должны быть нули. Я полагал, что, отметив биты, установлю их в единицу, а в результате закрыл все программирование. Пришлось исправить свою ошибку.

Биты конфигурации записались, я сбрасываю их и читаю. Появляется надежда, что вопрос с программированием решен.

Но первые «мягкие» грабли, на которые я наступаю – программа ведет себя не так, как я от нее ожидаю. Я загружаю в программатор файл, созданный по версии, написанной на языке С. В итоге, я могу записать в контроллер все нули или все единицы. Но стоит мне попытаться изменить в буфере хотя бы один байт, используя возможность редактирования в меню **Edit ➡ Edit Buffer enabled** (Редактирование ➡ Буфер редактирования доступен), чтобы записать что-то другое, как контроллер полностью обнуляется. С этой проблемой я долго не могу разобраться, поскольку не понимаю, вина ли это программы, или я что-то делаю не так. Я работаю с программой впервые и, судя по ошибке с установкой бит конфигурации, могу сделать множество ошибок. Конечно, все попытки обойти проблему успехом не увенчались.

Устав от проб и ошибок, я решаю изменить подход – если я не смогу запрограммировать контроллер, то сама микросхема,

которую я так берегу, мне будет не нужна. Либо я смогу ее запрограммировать, либо выброшу сразу.

Я решаюсь сделать попытку программировать ее как микросхему другого типа. Удачной оказывается попытка программирования **PIC16X84**. И я решаю, что именно так я буду ее программировать, а биты конфигурации запишу под эгидой PIC16F628. Это неудобно, но не покупать же дорогой программатор!

Итак, я могу записать в микросхему что-то кроме нулей и единиц.

Я загружаю HEX-файл, созданный программой, написанной на языке С, используя на инструментальной панели кнопку **Read Program Memory (FLASH)** (Прочитать память программы).

Вторые «мягкие» грабли выглядят так – вся программа не занимает и двух строк в буфере. Я в это не верю. Поскольку файл имеет расширение HEX, я пытаюсь просмотреть его HEX-редактором и не вижу ничего, кроме символьной записи. Тогда я открываю файл блокнотом:

```
:10000000830100308A00042820308400383000D201D
:10001000830139 2B04068001840A0406031D0A2883
:020020000034AA
:1004E20083018C1E712A1A0808008301B700831247
:1004F20003130C1E782A370899000800F401F5014D
:10050200 0310F30CF20C031C8D2A7008F407710817
:10051200318710AF5070310F00DF10D7208730448
:100522000319003 4812A830185010 7309F00831655
:10053200 8501FE30860090308312980006308316C3
:100542009800683099008312061083169B011C14D0 и т. д.
```

Признаться, не ожидал. Пришлось углубиться в руководства и понять, что этот файл предназначен для загрузки в программатор и формат файла имеет вид:

```
:BBAAAATTHHH… HHCC
```

Где BB – количество байт в строке файла, AAAA – адрес записи данных, TT – указатель типа строки (00 – данные, 01 – конец файла, 02 – адрес сегмента, 04 – линейный адрес), HHH…HH – собственно данные, а CC – контрольная сумма строки.

И данных в файле, конечно, больше, чем я получаю в буфере. В чем причина? После того как я понял, что после первых строк идет безусловный переход по адресу, находящемуся в конце программной области, а первый адрес после перехода лежит вне буфера, стало понятно, почему я не получил в буфере ничего интересного.

Первой приходит мысль исправить адреса в файле HEX. Но это потребует изменения контрольных сумм, которые придется либо рассчитывать на калькуляторе, либо писать программу пересчета. Сущность же проблемы состоит либо в том, что файл подготовлен под загрузку в программатор, работающий с MPLAB, либо в том, что срок действия демонстрационной версии компилятора языка С закончился.

В размышлениях – купить компилятор (каким-то образом) или найти знакомых, работающих с микроконтроллерами и «напроситься в гости», чтобы поработать некоторое время на чужом компьютере (меня эти походы изрядно «притомили»), я решаю поступить иначе.

Делаю последнюю попытку – компилирую программу в MPLAB для микросхемы PIC16F627A, которая имеет объем памяти программы вдвое меньше, чем моя PIC16F628A.

Теперь, загружая HEX-файл в программу PonyProg2000, я вижу больше пары строк в буфере. Записываю этот файл (для микросхемы PIC16X84), используя кнопку **Write Program Memory (FLASH)** (Записать память программы). Записываю биты конфигурации (для микросхемы PIC16F628). Проверяю в меню **Command ➡ Verify All** (Команда ➡ Проверить все) и получаю подтверждение правильности записи, затем переношу микросхему на макетную плату, где собран интерфейс и установлены светодиоды.

С этого момента опять начинаются «железные» грабли. Ничего у меня, естественно, не работает.

Первое, что следует сделать, – определиться, что у меня работает, а что нет. Уверен ли я, что программатор у работает? Нет. Уверен ли я, что программа написана правильно? Нет. И т.д. Я понимаю, что «нет» – это самое определенное, что я получил к настоящему моменту.

Тогда я решаю последовательно удалять эти «нет». В Интернете нахожу пример программирования на языке С для микроконтроллеров PIC – простая программа, которая заставляет светодиод мигать.

```
#include "pic16f62xa.h"

#define bitset(var,bitno) ((var) |= 1 << (bitno))
#define bitclr(var,bitno) ((var) &= ~(1 << (bitno)))

main() {
   unsigned int k;
   CMCON = 0x07;             //Компараторы выключены.
   TRISA = 0b11111110;          //RA0 выход.
repeat:
   for (k=0; k<45000; k++); //"Пустой" цикл для временной
задержки.
   bitset(PORTA, 0);         //Выставить на RA0 высокий уровень.
   for (k=0; k<45000; k++);
   bitclr(PORTA, 0);         //Выставить на RA0 низкий уровень.
   goto repeat;          //Повторить ещё раз.
   }
```

Я компилирую ее и программирую микросхему. После установки на макетную плату и включения питающего напряжения я получаю первое «да». Светодиод мигает. По меньшей мере, программатор работает.

Теперь я хочу проверить работу конвертера RS232–RS485. Удобнее всего было бы использовать осциллограф. Но у меня нет ничего кроме мультиметра с набором обычных функций измерения напряжения, тока, сопротивления.

Кое-что я, все-таки, проверить могу. Для этого мне потребуется программа работы с COM-портом. Можно написать на любом из языков программирования то, что работало бы с COM-портом. Но это займет время (позже это приходится сделать). Нахожу в Интернете программу под названием RS232Pro, в которой я смогу 25 дней поработать бесплатно. Есть, правда, предложение зарегистрировать программу, но сайт, где предлагается ее зарегистрировать, отсутствует. Думаю, 25 дней – это срок, когда я либо получу положительные результаты, либо откажусь от всего.

Проверяю, включив мультиметр, на измерение постоянного напряжения на пределе 20 В входные напряжения от COM-порта. Некоторое время уходит на то, чтобы понять, что нужно щелкать кнопку **RTS OFF** в программе RS232Pro, но, наконец, я делаю и этот решительный шаг. Затем отправляю в порт последовательность символов 12345, и вижу, что у меня меняется напряжение на выводе TXD. Меньше, но меняется напряжение на выводе 3 микросхемы MAX1483. Хотелось бы проверить линию RS485.

Для этой цели я пытаюсь передать простой текстовый файл с помощью программы RS232Pro, встав мультиметром на линию (в режиме измерения постоянного напряжения на пределе 20 В). Напряжение заметно меняется. Это убеждает меня, что конвертер, может быть, и не совсем правильно, но работает.

Порт и модуль я настраиваю на скорость 9600, 8 бит данных, 1 стоповый бит без бита четности. Первым делом, я упрощаю программу релейного модуля, оставив только один символ R на прием, который сразу зажигает светодиод. Модуль не работает, но обращение к документации на микросхему контроллера PIC16F628A заставляет меня задуматься в правильности выбора скорости. Я проверяю, меняя значение регистра SPBRG, разные скорости (соответственно, каждый раз перепрограммируя контроллер), пока не зажигается светодиод. Скорость оказывается равной 2400 (я даже не уверен, что ее не следует снизить до 1200). Модуль начинает реагировать на команды, передаваемые с компьютера.

В данный момент мне кажется, что дальше все образуется само собой, и модуль заработает. Но получается не совсем так. Мне даже удается получить ответ от модуля на запрос о состоянии реле. Ответ немного странный, но, хотя бы какой-то, он есть. Теперь беда другая. Одно выполнение команды, один ответ на запрос о состоянии реле, и модуль «виснет». Он не реагирует больше на команды до момента включения после сброса питания.

Я человек терпеливый, но и мое терпение истощается по мере продвижения к выполнению поставленной задачи. Главная

беда – я не уверен, что программа RS232Pro подходит для работы. По этой причине отправляюсь к знакомым, чтобы написать программку для работы с модулем. Думаю, я так надоел им, что они дарят мне старенькую версию Visual Basic (не совсем полную), которую кто-то из них получил на презентации во время одной из конференций. Мне важно, что версия работает.

Итак, я создаю заготовку для среды программирования (о чем речь пойдет в конце этой части книги), куда добавляю три кнопки: одну – для отправки команд, другую – для получения команд и последнюю – для того, чтобы полученные команды отобразить.

Используя оба варианта работы с COM-портом, я постепенно начинаю понимать, что после получения команды запроса состояния реле модуль начинает передавать ответ, но одновременно читает все, что есть на линии RS485. Прочитав же обращение к релейному модулю, свой номер модуля, он пытается реагировать на эти события. *То есть, при передаче модулем в сеть каких-либо данных нужно дать время на передачу – это во-первых, и отключить на время передачи приемник – это во-вторых.*

Не с первого раза получилась и запись состояния реле в EEPROM – скоро сказка сказывается, да не скоро дело делается.

Вот такие впечатления, которыми я обещал поделиться, остались у меня после посещения магазина и попытки претворить в жизнь компьютерную версию разработанного модуля.

Когда я написал, что отправляюсь в магазин, и поделюсь впечатлениями, я и сам не ожидал, что впечатлений будет так много. Конечно, отсутствие опыта, не совсем подходящие средства разработки, разница между тем, что нарисовано на бумаге, и тем, что будет создано на основе этого рисунка – все так, но... Я не ожидал, что столько «граблей» окажется на пути реализации достаточно простой конструкции.

Однако продолжим создание запланированных модулей.

Схема и программа модуля приема ИК-команд

Первый вопрос – зачем нам нужно что–либо, имеющее отношение к инфракрасным кодам?

Домашняя аппаратура – телевизоры, музыкальные центры и т.п. – управляется с помощью пультов дистанционного управления, излучающих команды в ИК-диапазоне. Чтобы управлять аппаратурой с компьютера (программно), нам потребуется излучатель ИК-кодов – модуль, который по команде компьютера будет излучать необходимые ИК-команды. Для работы этого модуля нам потребуется еще и считыватель ИК-кодов. Кроме того, пора подумать об устройствах управления в системе, причем хотелось бы иметь нечто достаточно дешевое.

Одним из устройств управления, как мы решили, станет компьютер. Можно подумать о создании устройства управления с использованием клавиатуры: нажатие клавиши отправляет в системную сеть команду управления.

Но и у компьютера, и у клавишного модуля есть небольшой недостаток: их удобно держать на стене или на столе, но не на кресле, где мы проводим достаточно много времени. Вопрос о клавишном модуле управления пока отложим и рассмотрим возможность управления с помощью старого пульта от видеомагнитофона или телевизора, которые давно отправились бы на свалку, если бы не завалялись на полке. В этом случае нам нужен **модуль приема инфракрасных кодов**.

Что собой представляют инфракрасные коды, излучаемые пультами управления?

Не вдаваясь в теоретические тонкости, можно сказать так: когда на пульте управления, положим, телевизором нажимается клавиша, установленный в нем светодиод (ИК-диапазона) начинает мигать. При этом он воспроизводит последовательность вспышек с некоторой частотой (от 20 до 400 кГц) и пауз, которые в совокупности и есть код управления. Каждая клавиша имеет свой набор вспышек и пауз. Клавиши разных пультов излучают разные коды управления, частота

(несущая частота) вспышек также может различаться. В качестве примера приведу структуру кодов управления фирмы Sony:

```
Technical Info
Code length (длина кода): 12 bits (15 bits для некоторых
функций видеокамеры)
Carrier (несущая):   40kHz
+------+
Header (Заголовок):   |  |
              +  +--+..
              __4T_____T


           +------+
1 кодируется:   |  |
+  +--+..
          __2T_____T


          +--+
0 кодируется:   |..|
          +..+--+..
          _T__T

T = 550us приблизительно

Пауза между данными: 25ms

data: hhhhxxxxxxyyyyyy
      ^            ^
      MSB          LSB

xxxxxx command (команда)
yyyyyy address (номер аппарата)
```

Таким образом, сначала идет заголовок длиной в 2,75 мс (инфракрасный свет мигает с частотой 40 кГц), затем пауза в 0,55 мс и код, в качестве которого для простоты рассмотрим последовательность 1001. Он будет соответствовать вспышке в 1,1 мс, паузе в 0,55 мс, вспышке в 0,55 мс, паузе в 0,55 мс, вспышке в 0,55 мс, паузе в 0,55 мс, вспышке в 1,1 мс, паузе в 0,55 мс, паузе в 25 мс. Уф!

Как будут выглядеть коды других производителей? Скорее всего, иначе. Есть несколько рекомендаций по применению

ИК-кодов, а производители вольны придерживаться их или нет. Кодирование логической единицы может производиться переходом (фронтом) от паузы к вспышке, логического нуля обратным переходом. Заголовок может отсутствовать и т.д.

Многие универсальные пульты управления, запоминающие ИК-коды, фиксируют длительность посылок и пауз, ждут длинной паузы между посылками, означающей завершение команды, а затем сохраняют значения в формате собственного времени, которое зависит от тактовой частоты синхрогенератора. Если при воспроизведении кода они могут менять несущую частоту, то записывают служебную информацию, ее определяющую.

Как и в предыдущем проекте, вначале приведу схему (рис. 1.45) и программу.

В табл. 1.4 перечислены все необходимые элементы.

Таблица 1.4. Спецификация модуля приема инфракрасных кодов

№	Обозначение	Изделие	Количество	Цена (р.)	Примечания
1	DD1	MAX1483	1	96	
2	DD2	PIC16F628A	1	100	Установить на панельку
3	DD3	LM2936-Z5	1	68	
4	DD4	TSOP17 (16)	1	10	
5	VD1	АЛ307	1	3	
7	R1	1 кОм 0,25	1	1	
8	R2	12 кОм 0,25 Вт	1	1	
9	R3	100 Ом 0,25 Вт	1	1	
10	R4–R7	10 кОм 0,25 Вт	4	4	
11	C1, C2	0,1 мкФ	2	6	
12	C3	100,0 мкФ 16 В	1	5	
13	C4	4,7 мкФ 5В	1	3	
14	X1	Клеммник 4 к	1	3	
15	S1	Перекл. 2 пол., 4 н	1	5	
16	Socket	DIP18	1	21	Панелька

Для обращения к модулю с запросом принята новая ИК-команда – `Cxx$0S` (аналогично команде запроса статуса релейного модуля).

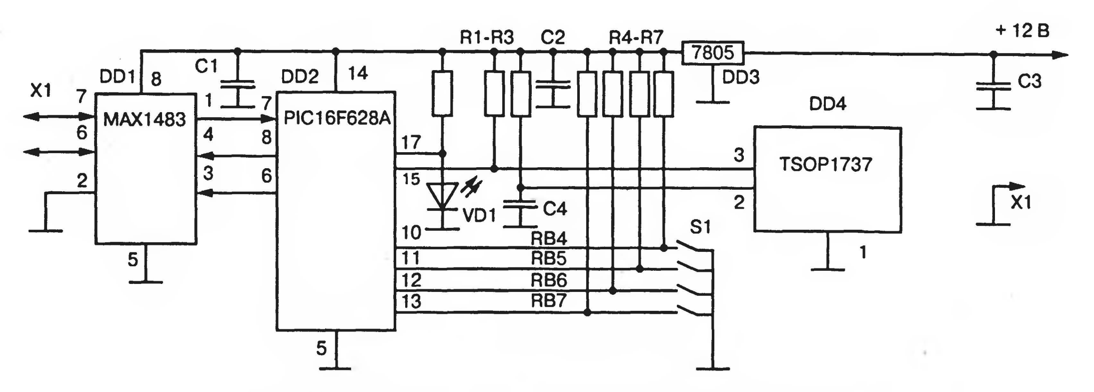


Рис. 1.45. Принципиальная схема модуля приема инфракрасных кодов

Ответ модуля – Cxxkkk, если команда пришла, и Cxx#ff, если новая ИК-команда не приходила.

Здесь xx, как и ранее, – двухбайтовый адрес модуля, kkk – три байта символов номера команды от 001 до 255, ff подтверждает отсутствие изменений за время опроса.

Программа модуля приема ИК-команд на языке С

Файл заголовка

```
#define MODULNAMESIM "C"
#define CMDSIM "$"

void putch(unsigned char);
unsigned char getch(void);
int init_comms();
int sim_num_adr();
int cmd();
```

Файл программы

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_rec_fnl.h"

unsigned char input;     // Считываем содержимое приемного
регистра.
unsigned char MOD_SIM1;     // Первый символ адреса модуля.
unsigned char MOD_SIM2;     // Второй символ адреса модуля.
unsigned char IRSIM1;
unsigned char IRSIM2;
unsigned char IRSIM3;
unsigned char command_reciev [6]; // Массив для полученной
команды.
int PHOTOCOME = 0;       // Флаг активности фотоприемника.
int MOD_ADDR;          // Заданный адрес модуля, как число.
unsigned char IRCOMMAND = 0;
int MOD_ADDR;          // Заданный адрес модуля, как число.
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
int MOD_NUM;           // Полученный адрес модуля, как число.
int i;
```

```
int k;
int l;
int m;

unsigned char getch()
{
   while((!RCIF)&(RB3 == 1)) // Устанавливается, когда
регистр не пуст.
      continue;
   return RCREG;
}

void putch (unsigned char byte)    // Вывод одного байта.
{
   while(!TXIF)    // Устанавливается, когда регистр пуст.
      continue;
   TXREG = byte;
}

int init_comms()    // Инициализация модуля.
{
PORTA = 0x00;
CMCON = 0x7;        // Настройка портов A и B.
TRISA = 0x00;
TRISB = 0xFE;
PCON = 0x8;         // Тактовая частота 4 МГц.
RCSTA = 0b10010000;       // Настройка приемника.
TXSTA = 0b00000110;       // Настройка передатчика.
SPBRG = 0x68;          // Настройка режима приема-передачи.
INTCON = 0x0;          // Запретить прерывания.
RB0 = 0x0;          // Выключим передатчик драйвера RS485.
PIE1 = 0x0;         // Настройка таймера 1, запрет прерывания.
T1CON = 0x0;        // Выбор внутреннего генератора, бит 1 в
ноль.
TMR1H = 0x00;
TMR1L = 0x00;
IRCOMMAND = 0x0;
            // Определим номер модуля.
MOD_ADDR = PORTB;      // Номер модуля в старших битах.
MOD_ADDR=MOD_ADDR>>4;     // Сдвинем на четыре бита.
}
            // Преобразуем символьный адрес в число.
int sim_num_adr()
```

```
{
   sim_end_num = 0x0;
   MOD_SIM1 = command_reciev [1]; // Первый символ номера.
   MOD_SIM2 = command_reciev [2]; // Второй символ номера.
   MOD_SIM1 = MOD_SIM1 - 0x30;
   MOD_SIM2 = MOD_SIM2 - 0x30;
   sim_end_num = MOD_SIM1*0x0A + MOD_SIM2;
   return sim_end_num;
}

int ir_cmd ()
{
   int b = 0;
   CREN = 0;
   while (RB3 == 0) continue;      // Ждем окончания
заголовка.
   for (b=0; b<8; b++)          //Обработаем наши импульсы.
   {
      while (RB3 != 0)
      continue;        // Дождемся импульса.
         time ();      // Включаем таймер 1.
         while (!TMR1IF);   // Дождемся такта.
         T1CON = 0x0;    // Выключаем таймер.
         TMR1IF = 0x0;      // Сбросим флаг.
         if (RB3 == 0)      // Если низкий уровень, то "1".
         {
         IRCOMMAND = IRCOMMAND +1;    // Запишем это.
         time ();     // Включаем таймер 1.
         while (!TMR1IF);   // Дождемся такта.
         T1CON = 0x0;    // Выключаем таймер.
         TMR1IF = 0x0;      // Сбросим флаг.
         } else       // Высокий уровень, значит "0".
            {
            IRCOMMAND = IRCOMMAND; // Запишем это.
            }
      if (b<7) IRCOMMAND = IRCOMMAND<<1;    //Сместимся влево
на бит.
   }
      PHOTOCOME = 0x0;
      RA0 = 0x0;
      RA1=0x1;
   for (m=0; m<4; ++m)
   {
```

```
      for (l=0; l<10000; ++l);
   }
   CREN = 1;
   RA1 = 0;
}

int cmd()
{
   CREN = 1;
   input = getch();
   switch (input)
   {
      case "C":       // Если обращение к фото модулю.
      for (i=1; i<6; ++i)      // Запишем команду в массив.
         {
         input = getch();
         command_reciev [i] = input;
         }
      MOD_NUM = sim_num_adr();     // Чтение из сети.
      if (MOD_NUM != MOD_ADDR) break;    // Если не наш адрес.
      else
         if (command_reciev [3] = "$")  // Если команда.
          {
            if (command_reciev [5] = "S") ir_stat();
          }
      default: break;
      }
}

void time()          // Таймер 1 для чтения ИК.
{
   TMR1H = 0xFD;       // Установка числа циклов до
переполнения.
   TMR1L = 0x43;       // FFFF минус 700 (2BCh).
   T1CON = 0x1;    // Включение таймера.
}

int ir_stat()
{
int ircom;
int ircom1;
int ircom2;
int ircom3;
```

```
command_reciev[0] = "C";
command_reciev[1] = MOD_SIM1+0x30;
command_reciev[2] = MOD_SIM2+0x30;

ircom = IRCOMMAND;    // Преобразуем команду в символы.
ircom1 = ircom/0x64;
IRSIM1 = ircom1 + 0x30;
ircom2 = (ircom - ircom1*0x64)/0xA;
IRSIM2 = ircom2 + 0x30;
ircom3 = (ircom - ircom1*0x64 - ircom2*0xA);
IRSIM3 = ircom3 + 0x30;

if (IRCOMMAND == 0)
{
command_reciev[3] = "#";
command_reciev[4] = "f";
command_reciev[5] = "f";

   CREN =0x0;       // Запрещаем прием.
   RB0 = 0x1;       //Переключим драйвер RS485 на передачу.
   TXEN = 0x1;      // Разрешаем передачу.
   for (i=0; i<6; ++i)  putch(command_reciev[i]);
   for (i=0;i<1000;i++);     // Задержка для вывода.
   for (i=0; i<6; ++i) command_reciev [i] = " ";
   RB0 = 0x0;       // Выключаем драйвер RS485 на передачу.
   TXEN = 0x0;      // Запрещаем передачу.
   CREN =0x1;       //Разрешаем прием.

} else   // За время между двумя запросами пришла ИК
команда.
   {
   command_reciev[3] = IRSIM1;
   command_reciev[4] = IRSIM2;
   command_reciev[5] = IRSIM3;

   CREN =0x0;       // Запрещаем прием.
   RB0 = 0x1;       //Переключим драйвер RS485 на передачу.
   TXEN = 0x1;      // Разрешаем передачу
   for (i=0; i<6; ++i)  putch(command_reciev[i]);
   for (i=0;i<1000;i++);     // Задержка для вывода.
   for (i=0; i<6; ++i) command_reciev [i] = " ";
   RB0 = 0x0;       // Выключаем драйвер RS485 на передачу.
TXEN = 0x0;      // Запрещаем передачу.
```

```
        CREN =0x1;        // Разрешаем прием.
        }
    IRCOMMAND = 0x0;   // Мы передали команду, она больше не
нужна  .
}

void main(void)        // Начнем работать.
{
    init_comms();           // Инициализация модуля.
    for (i=0; i<6; ++i) command_reciev [i] = " ";
    command_reciev [0] = "C";
    PHOTOCOME = 0x0;
    RA0 = 0x00;
                // Прочитаем и преобразуем номер модуля.
    MOD_ADDR = PORTB;  // Номер модуля в старших битах.
    MOD_ADDR=MOD_ADDR>>4;      // Сдвинем на четыре бита.

start:    RA2 = 0x1;        // Чтобы видно, что модуль включен.
          /* Нет ИК-сигнала, проверим сеть */
        CREN =0x1;
        if (RCIF) cmd();
        if (!RB3)    // Появился ИК сигнал.
        {
        CREN =0x0;
        for (k=0; k<30;++k);  // Поставим задержку.
        if (!RB3)    // ИК сигнал не пропал?
        {
            PHOTOCOME = 0x1;
            RA0 = 0x01;
                  // Обработаем ИК команду.
            ir_cmd ();
            PHOTOCOME = 0x0;
            RA0 = 0x00;
            goto start;
        }
        } else
        {
        PHOTOCOME = 0x0;
        RA0 = 0x00;
        CREN =0x1;
        }
    goto start;
}
```

HEX-файл для загрузки в программатор

```
:10000000830100308A000428203084004C300D2009
:100010008301102A04068001840A0406031D0A28AD
:020020000034AA
:1002D8008301861D71298C1E6C291A080800FD30BF
:1002E80083018F0043308E009001900A080083013B
:1002F800CB00831203130C1E7D294B0899000800BC
:10030800F401F5010310F30CF20C031C9229700898
:10031800F40771080318710AF5070310F00DF10DC1
:1003280072087304031900348629830 1B901BA01DC
:100338003C08A4003D08A500D030A407A5070A3052
:10034800F200F3012408F000F10184212508740764
:10035800B90075080318750ABA00F1003908F000E9
:100368000800830185010730 9F0083168501FE3050
:100378008600083 08E009030831298000630831 66D
:10038800980068309900 8B018312061083168C013F
:100398008312900 18F018E01A0010608A700A80111
:1003A8000430F000280DA80CA70CF00BD629080083
:1003B800830118166C21A6000B2AAD01AD0AAE0107
:1003C8002E08803AF000803070020630031 92D02A2
:1003D8000318FA296C21A6002D083B3E84008313DC
:1003E80026088000AD0A0319AE0AE429992170088D
:0803F800A9007108AA00280603
:1004000031D042A27082906031D08002430BE0006
:100410005330C000B62A2608433A031D0800E129DC
:10042000B521AD01AE011D2A2D083B3E840083138A
:1004300020308000AD0A0319AE0A2E08803AF00081
:100440008030700206300319 2D02031C142A433039
:10045000BB00AB01AC0105100608A700A8010430E1
:10046000F000280DA80CA70CF00B312A0515181662
:100470008C1ADC218619582A1812AF01B0013008F5
:1004800803AF000803070021E3003192F020318EA
:100490004D2AAF0A0319B00A3F2A8619362AAB0142
:1004A000AB0AAC0105145D22AB01AC010510362A84
:1004B000AB01AC01051018163 62A8301C101C20137
:1004C0001812861D612AC101C2018619652A73218D
:1004D0000C1C682A90010C108619752AA00A732139
:1004E0000C1C702A90010C10762A20084208803AD1
:1004F000F000803070020730031941020318822A8D
:100500000310A00DC10A0319C20A4208803AF00084
:1005100080307002083003194102031C652AAB01C8
:10052000AC0105108514B301B4013408803AF00021
```

```
:10053000803070020430031933020318B32AB1016A
:10054000B2013208803AF000A7307002103003196F
:1005500031020318AF2AB10A0319B20AA12AB30A59
:100560000319B40A952A1816851008004330830130
:10057000BB002408303EBC002508303EBD002008EA
:10058000C500C6016430F200F3014608F1004508D9
:10059000F000B1237408C7007508C8004708303E52
:1005A000A1006430F200F3014808F1004708F000B0
:1005B00084214608F1004508F0007408F002031C8D
:1005C000F1037508F1020A30F2000030F301B123A3
:1005D0007408C3007508C4004308303EA2000A3006
:1005E000F200F3014408F1004308F000842174088C
:1005F000C9007508CA006430F200F3014808F10030
:100600004708F00084214608F1004508F00074080E
:10061000F002031CF1037508F1024908F002031C03
:10062000F1034A08F1027008C1007108C2004108D4
:10063000303EA300A008031D632B2330BE006630AC
:10064000BF00C00018120614831698168312AD015D
:10065000AE012E08803AF000803070020630031997
:100660002D0203183D2B2D083B3E84008313000808
:100670007B21AD0A0319AE0A292BAD01AE012E086C
:10068000803AF00083307002E83003192D0203181D
:100690004D2BAD0A0319AE0A3F2BAD01AE012E085A
:1006A000803AF00080307002063003192D020318E2
:1006B000AA2B2D083B3E8400831320308000AD0A16
:1006C0000319AE0A4F2B2108BE002208BF002308E1
:1006D000C00018120614831698168312AD01AE01DD
:1006E0002E08803AF00080307002063003192D0287
:1006F0000318842B2D083B3E8400831300087B21C4
:10070000AD0A0319AE0A702BAD01AE012E08803A76
:10071000F00083307002E83003192D020318942B87
:10072000AD0A0319AE0A862BAD01AE012E08803A40
:10073000F00080307002063003192D020318AA2B36
:100740002D083B3E8400831320308000AD0A03193E
:10075000AE0A962B06108316981283121816A00163
:100760000800F601F11FBB2BF009F00A0319F10391
:10077000F1097617F61773088039F606F31FC72BB1
:10078000F209F20A0319F303F309C72BF601F40186
:10079000F50172087304031DD02BF001F101003440
:1007A0001F30F6040310F60AF20DF30D031CD32BD1
:1007B000F30CF20C73087102031DE02B7208700237
:1007C000031CE82B7208F0027308031C730AF10281
:1007D000F40DF50DF60BF61AD82BF61FF42BF409D1
```

```
:1007E000F40A0319F503F5097408F2007508F3001B
:1007F000761F0034F009F00A0319F103F1090034FF
:00000001FF
```

А для тех, кто не потерял интерес к процессу разработки собственных модулей и не спешит, я продолжу рассказ.

Итак. Какие требования мы предъявим к нашему модулю приемника инфракрасных кодов? Учтем, что основная задача модуля – принимать коды, которые мы назовем системными.

В качестве системных кодов, удобно было бы взять числа от 1 до 255 в формате выше разобранной структуры ИК-кода. На данном этапе примем этот вариант. Повторим еще раз, как будет выглядеть системный ИК-код. Вспышка с несущей частотой 37 кГц длительностью 2,2 мс (заголовок). Байт команды от 1 до 255 с соответствующими вспышками и паузами. Пауза 25 мс между командами.

Требования к модулю:

- модуль должен принимать системные ИК-команды с несущей частотой 37 кГц;
- модуль должен ответить на запрос центрального управляющего устройства (компьютера) по интерфейсу RS485, обозначив номер принятой ИК-команды.

Несколько слов о комплектующих элементах. Разумнее всего, с моей точки зрения, использовать фотоприемник TSOP1737 – рис. 1.46 – (если вы задействуете пульт с несущей частотой 37 кГц, в противном случае следует выбрать другой фотоприемник из этой серии). На рис. 1.47 схема его включения.

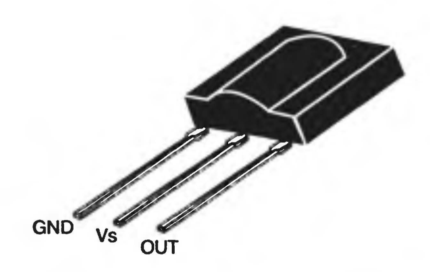


Рис. 1.46. Внешний вид и выводы фотоприемника

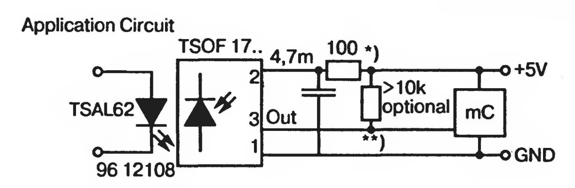


Рис. 1.47. Типовая схема включения фотоприемника TSOP

Конденсатор 4,7 мкФ желательно располагать непосредственно около фотоприемника, если между модулем и приемником несколько метров провода. Подобное может произойти, если вы не удовлетворитесь проверкой на макетной плате, а захотите воплотить модуль в жизнь и при этом решите убрать сам модуль подальше, а на виду оставить только фотоприемник. Фотоприемник невелик по габаритам, его можно приклеить с помощью двустороннего скотча на панель телевизора. Именно в этом случае я советую непосредственно к ножкам фотоприемника припаять конденсатор 4,7 мкФ. На выходе фотоприемника формируются сигналы (рис. 1.48): в отсутствие команд его выход в высоком состоянии; при наличии ИК-пульсаций (с частотой 37 кГц), он переходит в низкое состояние.

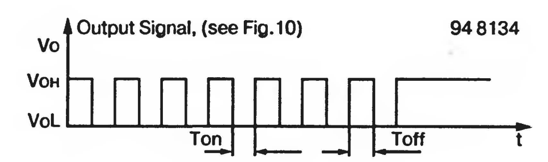


Рис. 1.48. Типовой вид сигнала на выходе фотоприемника

Здесь время Ton соответствует посылке ИК-импульсов, Toff – паузе.

Можно приступать к написанию программы.

Как и прежде, для установки заданного адреса используем четыре старших бита порта B (RB7–RB4). Для работы с сетью применим встроенный в микроконтроллер блок USART, а фотоприемник подключим к выводу RA6. Добавим еще светодиод для индикации наличия сигнала, который подключим

к выводу RA0. Поскольку остальные выводы портов нас не интересуют, отметим только, что в отличие от предыдущей конфигурации порта A – вывод RA6 – следует установить на ввод (в предыдущем варианте порт работал на вывод).

Основные блоки программы:

- инициализация модуля;
- ожидание активности приемника USART;
- ожидание активности от фотоприемника;
- обработка сетевой команды от компьютера;
- обработка ИК-команды от фотоприемника.

В предыдущей программе мы могли спокойно ждать активности в сети и ни о чем не заботиться, в этой программе нам приходится откликаться на два события – активность в сети и активность фотоприемника. Будем считать, что главное – это ожидание активности фотоприемника, и попробуем представить, что будет происходить при работе с реальной системой.

Каждое слово команды, состоящее из 6 байт, занимает (байт – 44 мкс) 0,264 мс. Положим, мы добавим 0,5 мс между командами.

Основное занятие компьютера – постоянный опрос модулей, поскольку программа должна выявлять события (изменение состояния модулей) и выполнять действия, соответствующие этим событиям. В данном плане модуль, который опрашивает и сеть, и фотоприемник, должен отслеживать изменения в обоих источниках. Пока модуль «слушает» сеть, он не «слышит» фотоприемник, и наоборот.

Вероятно, разумно было бы использовать механизм прерывания, хотя бы для работы по прерыванию с фотоприемником. Основанием стал бы тот факт, что в сети циркулируют запросы состояния, на которые модуль постоянно «отвлекается». При этом легко пропустить изменение состояния фотоприемника. Но для начала попробуем оценить ситуацию.

Мы определили длительность заголовка ИК-команды в 2,75 мс. Основное назначение заголовка – дать «понять» фотоприемнику, что далее последует команда.

Таким образом, изменение состояния фотоприемника с целью «привлечь внимание» контроллера модуля будет длиться около трех миллисекунд. За это время успеет пройти

около 10 (длительность слова команды в сети около 0,3 мс) слов команды. А это дает основания предполагать, что модуль успеет все «оценить» и перейдет к приему ИК-команды. В отсутствие прерываний он примет команду полностью, если это системная команда, прежде чем вернется к прослушиванию сети.

Однако скорость поступления ИК-команд едва ли будет превышать 0,3 с (интервал между двумя нажатиями на одну и ту же клавишу). То есть, за время между поступлениями команд у компьютера будет возможность послать около 1000 запросов. В итоге, если мы не планируем программу, которая будет нуждаться в 2000 запросов, а пользователь не будет играть на клавишах управления, как на рояле, особых проблем возникнуть не должно...

Мы не собираемся строить очень разветвленную сеть с большим количеством устройств. Но даже если бы мы этого захотели, то могли бы увеличить скорость работы в сети с 9600 бит/с (в наших рассуждениях) до 115 000 бит/с. Это изменение увеличило бы число возможных запросов до 12 000. Реальное время между командами я бы тоже оценил в 3–30 с, что позволило бы увеличить количество возможных сетевых запросов до 120 000. Если это не поможет, что ж, переделаем все полностью!

Итак, предварительная оценка, не исключаю, что ошибочная, позволяет нам считать возможной работу модуля без использования прерывания. На том и порешим.

Писать программу на ассемблере или на языке С? Я, пожалуй, пока не перестанет работать демонстрационная версия компилятора С, использую эту возможность.

Блок инициализации модуля

Не мудрствуя лукаво, поправим и используем в этом блоке код предыдущего модуля.

```
int init_comms()     // Инициализация модуля.
{
PORTA = 0x0;         // Настройка портов А и В.
CMCON = 0x7;
TRISA = 0xC0;             // Здесь только RA6 на ввод.
TRISB = 0xF6;
```

```
RCSTA = 0x90;          // настройка приемника.
TXSTA = 0x4;         // настройка передатчика.
SPBRG = 0x16;          // настройка режима приема-передачи.
INTCON = 0;          // запретить прерывания.
PORTB = 0;           // Выключим передатчик драйвера RS485.
}
```

Блок обработки команды фотоприемника

Пока RA6 в высоком состоянии («1»), ничего не надо делать. Проверим состояние USART. Если обращение идет не к нашему модулю, делать ничего не надо, вернемся к определению состояния RA6.

Я, как мне кажется, весьма рассудительно подошел к этому блоку программы. Но сомнения остаются. Я не уверен, будет ли она вообще работать, переключаясь с прослушивания от одного источника к другому.

Я, пожалуй, выделю блок и проверю его до написания остальной части программы. Не забудьте установить нужное состояние конфигурации (3F1Ah) по адресу 2007h!

Мне понадобится **файл заголовка**:

```
#define MODULNAMESIM "C"
#define CMDSIM "$"

void putch(unsigned char);
unsigned char getch(void);
int init_comms();
int sim_num_adr();
int cmd();
```

и **файл программы**:

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_rec.h"

unsigned char input;      // Считываем содержимое приемного
регистра.
unsigned char MOD_SIM1;  // Первый символ адреса модуля.
unsigned char MOD_SIM2;  // Второй символ адреса модуля.
```

```
unsigned char COMMAND;        // Символ команды.
int MOD_ADDR;            // Заданный адрес модуля, как число.
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
int MOD_NUM;          // Полученный адрес модуля, как число.
int PHOTO;
int PHOTOCOME = 0;
int FLAG = 0;            // Временная переменная.

unsigned char getch()
{
   while(!RCIF);        // Устанавливается, когда регистр не
пуст.
      continue;
   return RCREG;
}

              // Вывод одного байта.
void putch(unsigned char byte)
{
   PORTB = 1;       // Переключим драйвер RS485 на передачу.
   TXEN = 1;    // Разрешаем передачу.
   while(!TXIF)     // Устанавливается, когда регистр пуст.
      continue;
   TXREG = byte;
}

int init_comms()    // Инициализация модуля.
{
PORTA = 0xC0;            // Настройка портов A и B.
CMCON = 0x7;
TRISA = 0xC0;
TRISB = 0xF6;
RCSTA = 0x90;            // Настройка приемника.
TXSTA = 0x4;         // Настройка передатчика.
SPBRG = 0x16;            // Настройка режима приема-передачи.
INTCON=0;          // Запретить прерывания.
PORTB = 0;          // Выключим передатчик драйвера RS485.
}
              // Получение и выполнение команды.
int cmd()
{
```

```
   input = getch();
}
              // Преобразуем символьный адрес в число.

int sim_num_adr()
{
   sim_end_num = 0;
   sim1_num = getch();      // Чтение первого символа номера.
   MOD_SIM1 = sim1_num;     // Сохраним первый символ.
   sim2_num = getch();      // Чтение второго символа номера.
   MOD_SIM2 = sim2_num;     // Сохраним второй символ.
   sim1_num = sim1_num - 0x30;
   sim2_num = sim2_num - 0x30;
   sim_end_num = sim1_num*0x0A + sim2_num;
   return sim_end_num;
}
                 // Начнем работать.
void main(void)
{
   init_comms();          // Инициализация модуля.
//Прочитаем и преобразуем номер модуля.
   MOD_ADDR = PORTB;      // Номер модуля в старших битах.
   MOD_ADDR=MOD_ADDR>>4;    // Сдвинем на четыре бита.

start: PHOTO = PORTA;            // Начинаем работать.
   if (PHOTO&0x40) PHOTOCOME = 0; // Нет ИК-сигнала.
   if (!(PHOTO&0x40)) PHOTOCOME = 1; // Появился ИК-сигнал.
while (PHOTOCOME == 1)      // Обработаем ИК команду.
{
   FLAG = 1 ;
break;
}
                 // Нет ИК-сигнала, проверим сеть.
input = getch();
while(input == MODULNAMESIM)
{
   FLAG = 0 ;
   break;
}

   goto start;
}
```

В файл input.txt запишем:

```
"R03$0N"
"C03$0S"
```

Перед началом проверки запустим RB4 **Set High** (Установить в высокое состояние), RB5 **Set High** и RA6 **Set High** на Stimulus Controller (я изменил адрес с 01 на 03, а фотоприемник на вводе RA6 перевел в пассивное состояние).

Переменная FLAG покажет, попадаем ли мы в нужное место программы. Впоследствии мы вместо нее будем вызывать подпрограммы обработки команды `cmd()` и ИК-команды `ir_cmd()`. Кнопками RA6 **Set High** и RA6 **Set Low** мы будем имитировать изменение состояния фотоприемника. Вроде бы работает. Хотя, признаюсь, получилось не сразу – я забыл задать слово конфигурации по адресу 2007h.

Перейдем к обработке ИК-команды (обработку команд сети, я думаю, возьмем из предыдущей программы, подправим, попробуем).

После получения заголовка и паузы в 0,55 мс мы можем ожидать прихода всего двух сочетаний импульс–пауза. Ноль – это импульс–пауза одинаковой длительности 0,55 мс. Единица – импульс–пауза, где импульс – 1,1 мс, а пауза – 0,55 мс.

Таким образом, нам нужны два интервала в 1,1 мс и 0,55 мс. Для получения интервалов можно воспользоваться таймерами или отсчитать их в пустом цикле. Я воспользуюсь второй возможностью.

Все команды, кроме переходов, выполняются за один машинный цикл (200 нс при частоте 20 МГц). Положим, что при частоте 4 МГц машинный цикл равен 1 мкс. Тогда нам нужно досчитать до 1100 для промежутка в 1,1 мс и до 550 для короткого промежутка.

Сравнивая эти промежутки времени с приходящими импульсами, мы можем определить, единицу или ноль получаем в ИК-команде. Более того, мы можем установить только один счетчик до 550 (для верности возьмем 560) для вычисления байта команды.

На практике при первой проверке я успевал нажать кнопку RA6, чтобы имитировать приход ИК-команды. Теперь не

успеваю. Слишком все быстро. Попробуем поискать, не может ли MPLAB помочь с симуляцией, как это было в случае с USART.

И действительно, как любил повторять один из моих знакомых, «Грамотная программа!». В разделе **Debugger ➡ SCL Generator** (Отладчик ➡ SCL генератор) находим **New Workbook** (Новая рабочая тетрадь). Щелкаем и открываем окошко, в котором можно настроить имитатор работы фотоприемника.

В очередной раз наступив на грабли (вместо того чтобы почитать описание), я начинаю понимать, что в первой колонке нужно задать текущее время, выделяя события. Для этого следует, щелкнув кнопку со стрелкой правее надписи **Time Units** (Единицы времени), выбрать **мкс** (us). Событием в данном случае будет изменение состояния вывода RA6 порта A, которое мы зададим, щелкнув большую клавишу **Click here to Add Signals** (Щелкните здесь, чтобы добавить сигналы). Из раскрывающегося меню выберем RA6 и получим еще одну колонку с этим именем. Теперь в колонку **Time (dec)** впишем десятичные значения текущего времени (мкс), а в колонку RA6 – состояния входа, соединенного с фотоприемником. Не следует забывать, что в отсутствие активности фотоприемник на выходе имеет высокий логический уровень («1»). Первый ИК-сигнал, с которым я хочу работать, выглядит так: 10000000.

Если вернуться к разговору об ИК-командах и выбранному решению по системным командам, то сигнал, поступающий с фотоприемника на вход RA6, должен выглядеть, как показано на рис. 1.49 и 1.50.

Значения, которые следует проставить, представлены в табл. 1.5.



Рис. 1.49. Схема сканирования бит ИК-команды

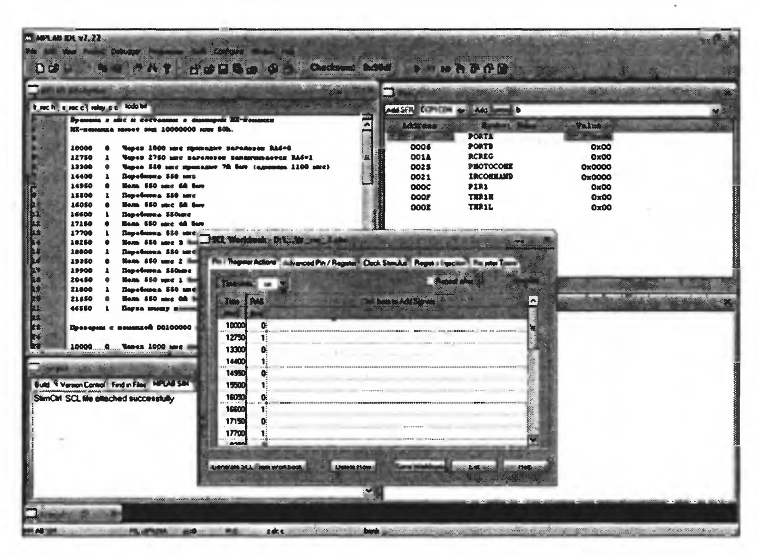


Рис. 1.50. Заполнение SCL Workbook

Таблица 1.5. Задание временных интервалов в сценарии ИК-команды

Time	RA6	Мои пометки (в книге)
10000	0	Через 1000 мкс приходит заголовок RA6 = 0
12750	1	Через 2750 мкс заголовок заканчивается RA6 = 1
13300	0	Через 550 мкс приходит 7-й бит (единица 1100 мкс)
14400	1	Перебивка 550 мкс
14950	0	Ноль 550 мкс 6-й бит
15500	1	Перебивка 550 мкс
16050	0	Ноль 550 мкс 5-й бит
16600	1	Перебивка 550мкс
17150	0	Ноль 550 мкс 4-й бит
17700	1	Перебивка 550 мкс
18250	0	Ноль 550 мкс 3-й бит
18800	1	Перебивка 550 мкс
19350	0	Ноль 550 мкс 2-й бит
19900	1	Перебивка 550мкс
20450	0	Ноль 550 мкс 1-й бит
21000	1	Перебивка 550 мкс
21550	0	Ноль 550 мкс 0-й бит
22100	1	Перебивка 550 мкс
47100	1	Пауза между командами 25 000 мкс

ИК-команда имеет вид 10000000 или 80h.

Теперь нужно сохранить Workbook и, щелкнув кнопку **Generate SCL From Workbook** (Генерировать SCL из рабочей тетради), сохранить .scl-файл. Для использования этой имитации ИК-команды в Simulus Controller необходимо щелкнуть кнопку **Attach** (Прикрепить), указав сохраненный прежде .scl-файл. После сохранения Workspace все необходимое будет появляться при загрузке проекта.

Спасибо создателям программы MPLAB, которые позаботились о возможности работы с портами ввода-вывода в разных их применениях!

Теперь я собираюсь проделать следующее: при активизации RA6 я перейду к функции обработки ИК-команды:

- пропускаем заголовок;
- пропускаем перебивку в 550 мкс;
- при активизации RA6 приходящим битом запустим «ожидалку» на 560 мкс (немного длиннее импульса в 550 мкс), после которой проверим состояние ввода RA6; если это ноль, бит «1», если единица, – бит «0» (на схеме сканирования это обозначено словом «Проверка»);
- если единица, еще раз включим «ожидалку», если ноль, не будем этого делать;
- запишем бит в переменную IRCOMMAND, сдвинем влево на одну позицию;
- все это проделаем с 8 битами ИК-команды.

Поскольку я человек ленивый, моя «ожидалка» самая простая:

```
for  (i = 0; i<560; ++i);      // Ждем-с
```

Было ли это наказанием за лень, или я не справился с настройками сценария, но, промучившись несколько часов, выяснил, что во время выполнения цикла `for` сценарий подхватывает циклы команды и успевает выполниться весь, прежде чем осуществляется выход из цикла ожидания. Я пробовал в качестве единиц измерения Time Units и циклы (cyc) и мкс (us) – не помогло.

Конечно, хорошо бы выяснить, в чем проблема, но единожды наказанный за лень, я решил из двух зол выбрать меньшее,

организовав «ожидалку» на таймере, как это и положено. Пока я, чертыхаясь, пытался оживить программу с циклом for, я заметил, что цикл while не мешает работе сценария.

В конечном счете, эта часть программы (в состоянии проверки) получилась следующей:

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_rec.h"

unsigned char input;      // Считываем содержимое приемного
регистра.
int PHOTOCOME = 0;    // Флаг активности фотоприемника.
int MOD_ADDR;         // Заданный адрес модуля, как число.
int IRCOMMAND = 0;

unsigned char getch()
{
   while(!RCIF)    // Устанавливается, когда регистр не пуст.
      continue;
   return RCREG;
}
              // Вывод одного байта.
void putch(unsigned char byte)
{
   PORTB = 1;      // Переключим драйвер RS485 на передачу.
   TXEN = 1;   // Разрешаем передачу.
   while(!TXIF)    // Устанавливается, когда регистр пуст.
      continue;
   TXREG = byte;
}

int init_comms()    // Инициализация модуля.
{
PORTA = 0xC0;           // Настройка портов A и B.
CMCON = 0x7;
TRISA = 0xC0;
TRISB = 0xF6;
RCSTA = 0x90;       // Настройка приемника.
TXSTA = 0x4;    // Настройка передатчика.
SPBRG = 0x16;       // Настройка режима приема-передачи.
INTCON=0;    // Запретить прерывания.
PORTB = 0;      // Выключим передатчик драйвера RS485.
```

```
PIE1 = 0;   // Настройка таймера 1, запрет прерывания.
T1CON = 0;     // Выбор внутреннего генератора, бит 1 в ноль.
}

int ir_cmd ()
{
   int b = 0;

   while ((PORTA&0x40)== 0) continue;// Ждем окончания
заголовка.
   for (b=0; b<8;++b)       // Обработаем наши импульсы.
      {
      while ((PORTA&0x40) != 0) continue;  // Дождемся
импульса.
         time ();         // Включаем таймер 1.
         while (!TMR1IF);       // Дождемся такта.
         T1CON = 0x0;        // Выключаем таймер.
         TMR1IF = 0x0;          // Сбросим флаг.
         if ((PORTA&0x40) == 0)   // Если низкий уровень, то
"1".
         {
         IRCOMMAND = IRCOMMAND +1;   // Запишем это.
         time ();         // Включаем таймер 1.
         while (!TMR1IF);       // Дождемся такта.
         T1CON = 0x0;        // Выключаем таймер.
         TMR1IF = 0x0;          // Сбросим флаг.
         } else      // Высокий уровень, значит "0".
         {
         IRCOMMAND = IRCOMMAND +0;   // Запишем это.
         }
         IRCOMMAND = IRCOMMAND<<1;   //Сместимся влево на
бит.
      }
// Уехали мы в смещениях на бит далеко, поэтому
         IRCOMMAND = IRCOMMAND>>1; // сместимся вправо на
бит.
         PHOTOCOME = 0;
}

int cmd()
{
   input = getch();
// switch (COMMAND = getch()) {
```

```
//     case "N": rel_on(REL_NUM);
//     break;
//     case "F": rel_off(REL_NUM);
//     break;
//     case "S": rel_stat(REL_NUM);
//     break;
//     }
}

void time()
{
   TMR1H = 0xFD;       // Установка числа циклов до
переполнения.
   TMR1L = 0xA7;       // FFFF минус 600 (258h).
   T1CON = 0x1;    // Включение таймера.
}

void main(void)    // Начнем работать.
{

   init_comms();       // Инициализация модуля.
//Прочитаем и преобразуем номер модуля.
   MOD_ADDR = PORTB;      // Номер модуля в старших битах.
   MOD_ADDR=MOD_ADDR>>4;     // Сдвинем на четыре бита.
start: if (PORTA&0x40) PHOTOCOME = 0;     // Нет ИК сигнала.
      if (!(PORTA&0x40)) PHOTOCOME = 1; // Появился ИК-
сигнал.
while (PHOTOCOME == 1)
{
                // Обработаем ИК команду.
   ir_cmd ();
break;
}
                // Нет ИК-сигнала, проверим сеть.
input = getch();
while(input == MODULNAMESIM)
{
   cmd ();
   break;
}
   goto start;
}
```

Все это получилось не сразу, но, вроде бы, заработало. Теперь осталось проверить программу с другими вариантами ИК-кода. Например, с командой 00100000, или 20h.

```
10000 0  Через 1000 мкс приходит заголовок RA6=0
12750 1  Через 2750 мкс заголовок заканчивается RA6=1
13300 0  Через 550мкс приходит 7й бит
13850 1  Перебивка 550 мкс
14400 0  Ноль 550 мкс 6й бит
14950 1  Перебивка 550 мкс
16050 0  Единица 1100 мкс 5й бит
17150 1  Перебивка 550 мкс
17700 0  Ноль 550 мкс 4й бит
18250 1  Перебивка 550 мкс
18800 0  Ноль 550мкс 3 бит
19350 1  Перебивка 550 мкс
19900 0  Ноль 550 мкс 2 бит
20450 1  Перебивка 550 мкс
21000 0  Ноль 550 мкс 1 бит
21550 1  Перебивка 550 мкс
22100 0  Ноль 550 мкс 0й бит
22650 1  Перебивка 550 мкс
47650 1  Пауза между командами 25000 мкс
```

С командой 10000011, или 83h:

```
10000 0  Через 1000 мкс приходит заголовок RA6=0
12750 1  Через 2750 мкс заголовок заканчивается RA6=1
13300 0  Через 550 мкс приходит 7-й бит (единица 1100 мкс)
14400 1  Перебивка 550 мкс
14950 0  Ноль 550 мкс 6-й бит
15500 1  Перебивка 550 мкс
16050 0  Ноль 550 мкс 5-й бит
16600 1  Перебивка 550мкс
17150 0  Ноль 550 мкс 4-й бит
17700 1  Перебивка 550 мкс
18250 0  Ноль 550 мкс 3-й бит
18800 1  Перебивка 550 мкс
19350 0  Ноль 550 мкс 2-й бит
19900 1  Перебивка 550мкс
20450 0  Единица 1100 мкс 1-й бит
21550 1  Перебивка 550 мкс
22100 0  Единица 1100 мкс 0-й бит
23200 1  Перебивка 550 мкс
48200 1  Пауза между командами 25000 мкс
```

И можно двигаться дальше. Предстоит добавить обработку запросов по системной сети, преобразование позиционного кода ИК-команды в символьный вид. Функцию обработки запросов по системной сети, подправив, я возьму из версии программы для предыдущего модуля.

Можно слегка изменить и сценарий имитации работы фотоприемника, добавив задание адреса модуля (пусть он пока остается «03») и подключение к порту А фотоприемника RA6 = 1.

Для этого в Workbook SCL Generator кнопкой **Click here to Add Signals** добавим еще три столбца: RB4, RB5 и PORTA, а в конце (как в табл. 1.5.) – информацию, представленную в табл. 1.6.

Таблица 1.6. Добавление к таблице временных интервалов

Time	RA6	RB4	RB5	PORTA	Мои пометки (в книге)
10000	0				Через 1000 мкс приходит заголовок, RA6=0
					И т.д.
47100	1				Пауза между командами 25000 мкс
5		1	1	40	Начальная инициализация адреса фотоприемника

Теперь при запуске программы будет задан адрес модуля, и фотоприемник установит вывод RA6 в «1».

В итоге получилась (не слишком аккуратная) следующая программа:

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_rec.h"

unsigned char input;      // Считываем содержимое приемного
регистра.
unsigned char MOD_SIM1;  // Первый символ адреса модуля.
unsigned char MOD_SIM2;  // Второй символ адреса модуля.
unsigned char IRSIM1;
unsigned char IRSIM2;
unsigned char IRSIM3;
int PHOTOCOME = 0;   // Флаг активности фотоприемника.
```

```
int MOD_ADDR;          // Заданный адрес модуля, как число.
int IRCOMMAND = 0;
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
int MOD_NUM;        // Полученный адрес модуля, как число.

unsigned char getch()
{
   while(!RCIF)    // Устанавливается, когда регистр не пуст.
      continue;
   return RCREG;
}
            // Вывод одного байта.
void putch(unsigned char byte)
{
   PORTB = 1;      // Переключим драйвер RS485 на передачу.
   TXEN = 1;    // Разрешаем передачу.
   while(!TXIF)    // Устанавливается, когда регистр пуст.
      continue;
   TXREG = byte;
}

int init_comms()    // Инициализация модуля.
{
PORTA = 0xC0;          // Настройка портов A и B.
CMCON = 0x7;
TRISA = 0xC0;
TRISB = 0xF6;
RCSTA = 0x90;          // Настройка приемника.
TXSTA = 0x4;        // Настройка передатчика.
SPBRG = 0x16;          // Настройка режима приема-передачи.

INTCON=0;        // Запретить прерывания.
PORTB = 0;          // Выключим передатчик драйвера RS485.
PIE1 = 0;        // Настройка таймера 1, запрет прерывания.
T1CON = 0;       // Выбор внутреннего генератора, бит 1 в ноль.

            // Определим номер модуля.
MOD_ADDR = PORTB;      // Номер модуля в старших битах.
MOD_ADDR=MOD_ADDR>>4;     // Сдвинем на четыре бита.
}
            // Преобразуем символьный адрес в число.
```

```
int sim_num_adr()
{
   sim_end_num = 0;
   sim1_num = getch();        // Чтение первого символа номера.
   MOD_SIM1 = sim1_num;       // Сохраним первый символ.
   sim2_num = getch();        // Чтение второго символа номера.
   MOD_SIM2 = sim2_num;       // Сохраним второй символ.
   sim1_num = sim1_num - 0x30;
   sim2_num = sim2_num - 0x30;
   sim_end_num = sim1_num*0x0A + sim2_num;
   return sim_end_num;
}

int ir_cmd ()
{
   int b = 0;

   while ((PORTA&0x40)== 0) continue;// Ждем окончания
заголовка.
   for (b=0; b<8;++b)         //Обработаем наши импульсы.
      {
      while ((PORTA&0x40) != 0) continue;   // Дождемся
импульса.
         time ();          // Включаем таймер 1.
         while (!TMR1IF);        // Дождемся такта.
         T1CON = 0x0;         // Выключаем таймер.
         TMR1IF = 0x0;           // Сбросим флаг.
         if ((PORTA&0x40) == 0)     // Если низкий уровень, то
"1".
         {
         IRCOMMAND = IRCOMMAND +1;    // Запишем это.
         time ();          // Включаем таймер 1.
         while (!TMR1IF);        // Дождемся такта.
         T1CON = 0x0;         // Выключаем таймер.
         TMR1IF = 0x0;           // Сбросим флаг.
         } else       // Высокий уровень, значит "0".
         {
         IRCOMMAND = IRCOMMAND +0;    // Запишем это.
         }
         IRCOMMAND = IRCOMMAND<<1;    //Сместимся влево на бит.
      }
// Уехали мы в смещениях на бит далеко, поэтому
      IRCOMMAND = IRCOMMAND>>1;    // сместимся вправо на бит.
```

```
    PHOTOCOME = 0;
}

int cmd()
{
    MOD_NUM = sim_num_adr();     //Чтение из сети (файла).
    if (MOD_NUM == MOD_ADDR)     // Если наш адрес модуля.
    {
      input = getch();
      if (input == "$")      // Если символ команды.
      {
      while ((input = getch())!= "S");  // Ждем.
      ir_stat();
      }
    }
}

void time()
{
   TMR1H = 0xFD;       // Установка числа циклов до
переполнения.
   TMR1L = 0xA7;       // FFFF минус 600 (258h).
   T1CON = 0x1;    // Включение таймера.
}

int ir_stat()
{
int ircom;
int ircom1;
int ircom2;
int ircom3;

   ircom = IRCOMMAND;    // Преобразуем команду в символы.
   ircom1 = ircom/0x64;
   IRSIM1 = ircom1 + 0x30;
   ircom2 = (ircom - ircom1*0x64)/0xA;
   IRSIM2 = ircom2 + 0x30;
   ircom3 = (ircom - ircom1*0x64 - ircom2*0xA);
   IRSIM3 = ircom3 + 0x30;
   putch("C");
   putch(MOD_SIM1);
   putch(MOD_SIM2);
   if (IRCOMMAND == 0)      // Команда не менялась.
```

```
    {
    putch("#");
    putch("f");
    putch("f");
    putch(0x0A);         // Только для вывода в файл!!!!!!
    } else    // За время между двумя запросами пришла ИК
команда.
        {
        putch(IRSIM1);
        putch(IRSIM2);
        putch(IRSIM3);
        putch(0x0A);     // Только для вывода в файл!!!!!!
        }
    IRCOMMAND = 0;  // Мы передали ИК команду, она нам  больше
не нужна.
}

void main(void)     // Начнем работать.
{
    init_comms();        // Инициализация модуля.
// Прочитаем и преобразуем номер модуля.
    MOD_ADDR = PORTB;  // Номер модуля в старших битах.
    MOD_ADDR=MOD_ADDR>>4;     // Сдвинем на четыре бита.
// Начинаем работать
start: if (PORTA&0x40) PHOTOCOME = 0;     // Нет ИК-сигнала.
        if (!(PORTA&0x40)) PHOTOCOME = 1; // Появился ИК-
сигнал.
while (PHOTOCOME == 1)
{
                    // Обработаем ИК-команду.
    ir_cmd ();
break;
}
                    // Нет ИК-сигнала, проверим сеть.
input = getch();
while(input == MODULNAMESIM)    // Если в сети обращение к
модулю.
{
    cmd ();              // Обработаем сетевую команду.
    break;
}
    goto start;
}
```

В настоящий момент меня уже одолевают сомнения. Пока активность в сети не мешает приему ИК-команд. Не будет ли она мешать, когда появится функция обработки запросов? Можно попытаться рассчитать это, но расчеты могут оказаться достаточно сложными. И не будет ли мешать прием ИК-команд системным запросам? Не стану считать, попробуем, посмотрим.

В крайнем случае, если не забуду, в основной программе системы при отсылке запроса о состоянии модуля ИК-приемника, сделаю паузу и при отсутствии ответа повторю запрос.

Есть и еще одно сомнение – не будет ли в реальных условиях неправильно считываться ИК-команда? Здесь тоже можно что-нибудь придумать. Например, повторять системную команду трижды, а в модуль добавить три считывания, принимая в качестве команды ту, которая совпадает дважды. На данном этапе я предпочту отложить решение проблем до момента их появления, хотя, как мне кажется, самое время обо всем позаботиться.

Работа выше приведенной программы происходит при input.txt такого вида:

```
"R03$0N"
"R01$0N"
"C03$0S"
```

Опция **Rewind Input** на странице **Uart1 IO** установлена. В результате имитация сетевых команд постоянно «крутится», принимая три команды, последняя из которых запрашивает состояние модуля фотоприемника.

Результат работы программы выводится, как и раньше, в файл output.txt:

```
C03#ff // Команда не менялась
C03131 // Пришла команда 131 (десятичная) или 10000011
двоичная
C03#ff // Команда не менялась после последнего запроса
C03#ff // Команда не менялась
C03#ff // Команда не менялась
```

По дороге я еще раз наступил на грабли. Внеся исправления в программу, я обнаружил, что она не желает работать. Во всяком случае, работать так, как мне хочется. Пытаясь понять, в чем дело, я много раз просматривал программу, пока не обнаружил, что один из операторов имеет тот же цвет, что и комментарий. Как это получилось? Я использую много однострочных комментариев (тоже следствие лени) и если в конце строчного комментария не нажат ввод при удалении промежутков между операторами программы, следующий воспринимается программой как продолжение комментария.

Чтобы частично развеять сомнения, я хочу переделать программу для тестирования:

- проверить, правильно ли будет работать программа, если после первой ИК-команды вторая придет через 50 мс;
- проверить, много ли запросов о состоянии модуля пропускается, если это происходит.

Для этого в папке, где лежат все проекты, я создам еще одну папку, которую обозначу, добавив _test к имени. Скопирую туда содержимое папки модуля приемника ИК-команд. Переименую все файлы, добавив _test (исключая файлы input и output). Затем открою этот проект в программе MPLAB, настрою, добавлю в сценарий второй ИК-код, счетчики и попробую ответить на два вопроса, которые задаю себе сейчас.

Переделки

В раздел переменных добавим (я использую глобальные переменные, чтобы их легче было отслеживать, поскольку локальные переменные до обращения к функциям не видны):

```
int COUNTER = 0;
```

А в основной программе – счетчик после вызова команды обработки сетевых команд:

```
   /* Нет ИК-сигнала, проверим сеть */
input = getch();
while(input == MODULNAMESIM) // Если в сети обращение к модулю.
```

```
{
      cmd ();      // Обработаем сетевую команду.
      ++COUNTER;
   break;
}
   goto start;
```

Результат эксперимента – за время прохождения двух ИК-команд (83h и 20h) счетчик досчитал до 5.

Файл input.txt, который «крутится» без перерывов выглядит так:

```
"R03$0N"
"R01$0N"
"C03$0S"  Обращение к модулю приемника ИК команд
"R03$0N"
"R01$0N"
"C03$0S"  Обращение к модулю приемника ИК команд
```

Файл output.txt получился следующим:

```
C03#ff Команда не обновлялась
C03131 Команда 131 (83h)
C03#ff Команда не обновлялась
C03#ff Команда не обновлялась
C03032 Команда 32 (20h)
```

Таким образом:

- ИК-команды не потерялись за непрерывно следующими сетевыми командами;
- ИК-команды прочитались правильно;
- между приходами ИК-команд запрос статуса правильно отображал отсутствие обновления.

Посмотрим, как это все выглядит во времени.

Первая ИК-команда приходит через 10 мс и заканчивается через 24 мс, вторая приходит через 60 мс и заканчивается через 73 мс после начала работы. Сетевые команды начинают поступать сразу после начала работы. Каждая сетевая команда занимает около 6,3 мс (имеет 60 бит, проходящих со скоростью ~9600 бит/с).

До начала ИК-команды при такой скорости должна пройти одна сетевая команда. За время считывания первой

ИК-команды ((10 мс – 6 мс) + 24 мс) = 28 мс пройдет 4 команды. После завершения первой и до конца второй пройдет 73 мс – 24 мс = 49 мс, что соответствует 7–8 командам. Таким образом, за все время пройдет 12–13 команд. То есть, файл input прочитается дважды или немного более. Что дает 4–5 обращений к модулю приемника ИК-команд. Файл output фиксирует 5 обращений, счетчик фиксирует 5 обращений.

Для большей уверенности посмотрим, как распределяются сетевые обращения:

- за время от начала работы до завершения считывания ИК-команды проходит 5 системных запросов, среди которых один к ИК-приемнику;
- в выходном файле этому соответствует один ответ об отсутствии обновления команд;
- следующая команда во входном файле это запрос к модулю;
- в выходном файле на втором месте стоит ответ о приходе команды 131 (к этому моменту считывание первой команды завершено);
- до завершения считывания второй команды проходит семь системных команд, среди которых два обращения к модулю приемника;
- в выходном файле два ответа – команда не обновлялась (предыдущая уже передана);
- следующая сетевая команда – запрос о состоянии модуля приемника, который следует сразу за завершением считывания ИК-команды, ответ – команда 32 в выходном файле.

Если я ничего не напутал, даже при короткой основной программе (центрального управляющего устройства), когда запросы о состоянии модуля приемника идут достаточно часто, ответ успевает доходить до адресата. Если учесть, что в тестовом варианте две команды проходят с интервалом в 36 мс, что в 10–30 раз быстрее реально отправляемых команд (интервал между двумя нажатиями на одну и ту же клавишу составляет от 0,3 до 1 с), то запас, я думаю, есть.

Конечно, мои подсчеты не страдают излишней продуманностью и точностью, но я на время решил успокоиться.

Отладка модуля

Прототип я делаю на той же макетной плате, на которой собирал релейный модуль. По этой причине я включаю фотоприемник на вход RB3. Для индикации приема ИК-команд дополнительно использую вывод RA0, к которому уже подключен светодиод. Когда устанавливается флаг прихода ИК-команды, светодиод включается. Флаг снимается – светодиод выключается.

```
if (!(RB3&0x01))
{
   PHOTOCOME = 1;
   RA0 = 0x01;
}
 else
{
   PHOTOCOME = 0;
   RA0 = 0x00;
}
```

Вот и очередные «грабли»!

Приоритет приема ИК-команд хорошо бы закрепить однозначно. Например, так:

```
{
   while((!RCIF)&(RB3 == 1))
      /* устанавливается, когда регистр не пуст */
      continue;
   return RCREG;
}
```

Я выделил добавленный фрагмент. У прототипа фотоприемник подключен к выводу RB3. Смысл добавленного фрагмента в том, что на сетевые запросы ответ будет, если модуль не занят приемом ИК-команд, то есть RB3 = 1.

Неплохо было бы защитить модуль от помех по RB3. При включении может «проскочить» короткий нулевой импульс (на чем я и споткнулся), и модуль перейдет в режим приема ИК-команды, которой нет. Поправил я это так:

```
start: if (RB3&0x01)   // Нет ИК сигнала.
   {
```

```
      PHOTOCOME = 0;
      RA0 = 0x00;
break;
   }
   if (!(RB3&0x01))        // Появился ИК сигнал.
   {
      for (k=0; k<30; ++k); // Поставим задержку.
      if (!(RB3&0x01))   // ИК сигнал не пропал?
      {
         PHOTOCOME = 1;
         RA0 = 0x01;
      }
   } else
      {
         PHOTOCOME = 0;
         RA0 = 0x00;
         break;
      }
```

Я добавляю задержку, а затем еще раз проверяю наличие активности фотоприемника. Короткий случайный импульс проскочит за время задержки, а длинный импульс команды приведет механизм считывания в действие. Помогло. Если в первый раз после включения питания (сразу или через несколько секунд) модуль «подвисал» на приеме ИК-команды, то после исправления перестал.

Итак, модуль получает команду. Возможно, распознает ее. Он даже передает ее по запросу через RS485. А что получится, если команда не единичная, то есть в том случае, когда идет поток одинаковых команд? Что будет, если команды не системные, а чужие? Не знаю.

Первое, что приходит в голову – запустить еще один таймер сразу после завершения считывания ИК-команды. И пока он не завершит отсчет времени больший, чем занимают 2–3 команды, не принимать ИК-коды. Попутно я пытаюсь воспроизвести код, аналогичный тому, что использовал в отладчике MPLAB, из программы WinLIRC, аппаратный модуль которой имеет излучающий светодиод.

Проходит два дня. Результаты не впечатляют. Коды прочитываются не слишком уверено. Я увеличиваю интервал сканирования с 600 до 700 мкс, но совсем не удовлетворен

стабильностью работы модуля. В отличие от релейного, модуль приема системных ИК-команд мне совсем не нравится. Он, вроде бы, работает, но...

В конечном счете, я меняю излучатель кодов. Использую оригинал – старый пульт от видеомагнитофона Sony. Меняю второй таймер на обычный цикл в команде считывания кодов (выделено в тексте программы), попутно решаю внести исправления в сдвиг на один бит. У меня получился лишний сдвиг, который я после получения ИК-команды убирал обратным сдвигом. После замены переменной IRCOMMAND с типа int на unsigned char я стал терять старший бит.

Замена всех этих «шатаний из стороны в сторону» оказалась простой (изменение выделено). Если я не ошибаюсь, этого достаточно.

```
int ir_cmd ()
{
   int b = 0;

   while (RB3 == 0)   continue; // Ждем окончания заголовка.
   for (b=0; b<8; b++)           //Обработаем наши импульсы.
      {
      while (RB3 != 0)
      continue;         // Дождемся импульса.
         time ();      // Включаем таймер 1.
         while (!TMR1IF);   // Дождемся сброса таймера.
         T1CON = 0x0;    // Выключаем таймер.
         TMR1IF = 0x0;       // Сбросим флаг.
         if (RB3 == 0)       // Если низкий уровень, то "1".
         {
         IRCOMMAND = IRCOMMAND +1;    // Запишем это.
         time ();          // Включаем таймер 1.
         while (!TMR1IF);   // Дождемся сброса таймера.
         T1CON = 0x0;    // Выключаем таймер.
         TMR1IF = 0x0;       // Сбросим флаг.
         } else        // Высокий уровень, значит "0".
         IRCOMMAND = IRCOMMAND;         // Запишем это.
   if (b<7) IRCOMMAND = IRCOMMAND<<1;        //Сместимся влево
на бит.
      }
      PHOTOCOME = 0x0;
```

```
    RA0 = 0x0;
    RA1=0x1;
    for (m=0; m<3; ++m)       // Перестанем принимать ИК-коды.
    for (l=0; l<10000; ++l);
  RA1 = 0x0;
}
```

Здесь на выводе порта A RA1 я использую еще один светодиод, чтобы видеть окончание команды.

В результате модуль хорошо распознает команды от пульта видеомагнитофона. Чувствительность высокая, распознавание стабильное (рис. 1.51).



Рис. 1.51. Прием ИК-кодов модулем

Не удовлетворяют меня только два момента:

- иногда во время паузы между приемами ИК-команд запрос по сети «подвешивает» модуль в нераспознанном месте (помогло сокращение времени этой паузы);
- коды с пульта плохо согласуются с моими ожиданиями.

Вот сравнение кодов, прочитанных с помощью WinLIRC в режиме распознавания, и кодов, которые транслирует модуль в ответ на запрос по сети RS485:

Номер канала 1.

Модуль возвращает номер команды 001 – 00000001 (C14001 – в ответ на запрос статуса).

7F2h – 11111110010 (инверсия 00000001101), без последних трех бит 0000001.

Номер канала 2.

Прочитано модулем 129 – 10000001

3F2h – 01111110010 (инверсия 10000001101), без последних трех бит 10000001.

Номер канала 3.

Прочитано модулем 65 – 1000001

5F2h – 10111110010 (инверсия 01000001101), без последних трех бит 1000001

Номер канала 6.

Прочитано модулем 161 – 10100001

2F2h – 01011110010 (инверсия 10100001101), без последних трех бит 10100001.

Номер канала 9.

Прочитано модулем 017 – 10001

772h – 11101110010 (инверсия 00010001101), без последних трех бит 00010001.

power.

Прочитано модулем169 – 10101001

2B2h – 1010110010 (инверсия 0101001101), без последних трех бит 0101001.

Команда включения канала 1 в записи WinLIRC – 7F2h, но я определяю единицу и ноль, как это описано в технической информации о кодах Sony, которой располагаю, а в WinLIRC сделано наоборот. Можно инвертировать код. Кроме того, я использую 8-битовый код, а оригинальный код – 11-битовый. Тоже не страшно, отбросим три последних бита. Смущающим меня обстоятельством служит то, что команды каналов 1 и 2 не совпадают, как я ожидал после превращений с инверсией и отбрасыванием. Аналогичная история с командой power. Пока я не понимаю, как это происходит.

Шутка калькулятора, который отбрасывает первые нули! Но понял я это позже.

Последнее, что я делаю, измученный борьбой с «собственной гениальностью», – проверяю работу модуля в заготовке под среду программирования. Модуль работает. Я использую два кода: номер включения канала 1 – для тестирования команды, а канала 2 – для включения лампы. Оба кода стабильно работают.

Думаю, пока следует остановиться. Основная задача – получить модуль считывания системных команд, которые стабильно распознавались бы системой, достигнута. При этом, как это и планировалось, используется старенький пульт от видеомагнитофона. А если что-то осталось непонятно, что ж... Будем надеяться, что это позже прояснится (рис. 1.52).

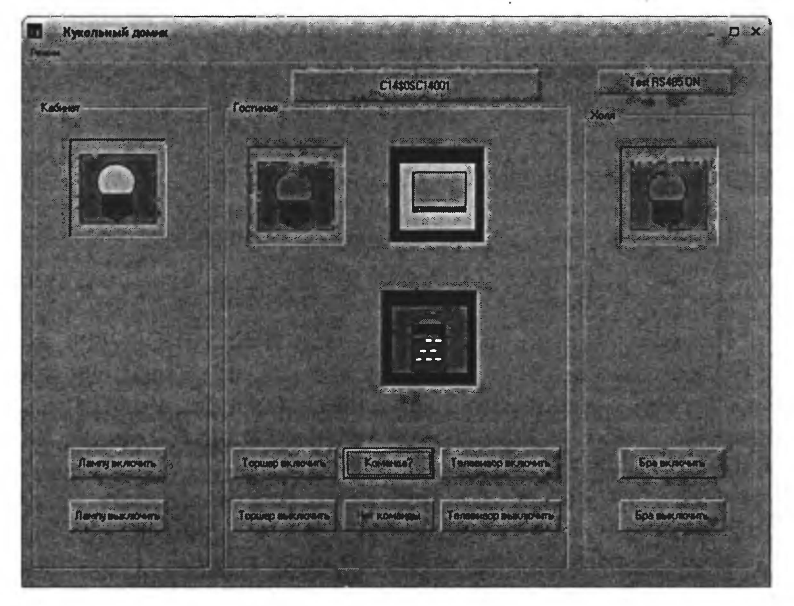


Рис. 1.52. Работа модуля с основной программой

Самым непонятным для меня оказалось то, что попытка использовать третий светодиод для индикации включенного модуля – я несколько раз вытаскивал микросхему из панельки без отключения питающего напряжения – успехом не увенчалась. Я попробовал разные варианты включения. Ничего не получилось! Мистика!?

В данный момент программа модуля приема системных кодов получилась такой.

Файл заголовка:

```
#define MODULNAMESIM "C"

unsigned char getch(void);
int init_comms();
int ir_cmd ();
int cmd ();
void time();
void time_cmd();
int ir_stat();
```

Файл основной программы:

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_rec_01.h"

unsigned char input;      // Считываем содержимое приемного
регистра.
unsigned char MOD_SIM1;   // Первый символ адреса модуля.
unsigned char MOD_SIM2;   // Второй символ адреса модуля.
unsigned char IRSIM1;
unsigned char IRSIM2;
unsigned char IRSIM3;
unsigned char command_reciev [6]; // Массив для полученной
команды.
int PHOTOCOME = 0;        // Флаг активности фотоприемника.
int MOD_ADDR;             // Заданный адрес модуля, как число.
unsigned char IRCOMMAND = 0;
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
int MOD_NUM;            // Полученный адрес модуля, как число.
```

```
int i;
int k;
int l;
int m;

unsigned char getch()
{
   while((!RCIF)&(RB3 == 1))   // Когда регистр не пуст.
      continue;
   return RCREG;
}
                 // Вывод одного байта.
void putch(unsigned char byte)
{
   while(!TXIF)        // Когда регистр пуст.
      continue;
   TXREG = byte;
}

int init_comms()    // Инициализация модуля.
{
PORTA = 0x00;
CMCON = 0x7;        // Настройка портов А и В.
TRISA = 0x00;
TRISB = 0xFE;
PCON = 0xFF;        // Тактовая частота 4 МГц.
RCSTA = 0b10010000;       // Настройка приемника.
TXSTA = 0b00000110;       // Настройка передатчика.
SPBRG = 0x68;          //Настройка режима приема-передачи 2400,
N,8,1.
INTCON = 0x0;          // Запретить прерывания.
RB0 = 0x0;          // Выключим передатчик драйвера RS485.
PIE1 = 0x0;         // Настройка таймера 1, запрет прерывания.
T1CON = 0x0;        // Выбор внутреннего генератора, бит 1 в
ноль.
TMR1H = 0x00;          // Обнулим таймер.
TMR1L = 0x00;
IRCOMMAND = 0x0;             /* Определим номер модуля */
MOD_ADDR = PORTB;      // Номер модуля в старших битах.
MOD_ADDR=MOD_ADDR>>4;     // Сдвинем на четыре бита.
}
              // Преобразуем символьный адрес в число.
int sim_num_adr()
```

```
{
   sim_end_num = 0x0;
   MOD_SIM1 = command_reciev [1];     // Первый символ номера.
   MOD_SIM2 = command_reciev [2];     // Второй символ номера.
   MOD_SIM1 = MOD_SIM1 - 0x30;
   MOD_SIM2 = MOD_SIM2 - 0x30;
   sim_end_num = MOD_SIM1*0x0A + MOD_SIM2;
   return sim_end_num;
}

int ir_cmd ()
{
   int b = 0;

   while (RB3 == 0)  continue; // Ждем окончания заголовка.
   for (b=0; b<8; b++)          //Обработаем наши импульсы.
      {
      while (RB3 != 0)
      continue;         // Дождемся импульса.
         time ();       // Включаем таймер 1.
         while (!TMR1IF);   // Дождемся сброса таймера.
         T1CON = 0x0;    // Выключаем таймер.
         TMR1IF = 0x0;      // Сбросим флаг.
         if (RB3 == 0)      // Если низкий уровень, то "1".
         {
         IRCOMMAND = IRCOMMAND +1;    // Запишем это.
         time ();          // Включаем таймер 1.
         while (!TMR1IF);        // Дождемся сброса.
         T1CON = 0x0;         // Выключаем таймер.
         TMR1IF = 0x0;           // Сбросим флаг.
         } else       // Высокий уровень, значит "0".
         IRCOMMAND = IRCOMMAND;        // Запишем это.

      if (b<7) IRCOMMAND = IRCOMMAND<<1;    //Сместимся влево
на бит.
      }
      PHOTOCOME = 0x0;
      RA0 = 0x0;
      RA1=0x1;
      for (m=0; m<3; ++m)       // Перестанем принимать ИК.
         for (l=0; l<10000; ++l);
      RA1 = 0x0;
}
```

```
int cmd()
{
   if (command_reciev [5] = "S") ir_stat();
}

void time()          // Таймер 1 для чтения ИК.
{
   TMR1H = 0xFD;       // Установка числа циклов до
переполнения.
   TMR1L = 0x43;       // FFFF минус 700 (2BCh).
   T1CON = 0x1;    // Включение таймера.
}

int ir_stat()
{
int ircom;
int ircom1;
int ircom2;
int ircom3;

   command_reciev[0] = "C";
   command_reciev[1] = MOD_SIM1+0x30;
   command_reciev[2] = MOD_SIM2+0x30;

   ircom = IRCOMMAND;     // Преобразуем команду в символы.
   ircom1 = ircom/0x64;
   IRSIM1 = ircom1 + 0x30;
   ircom2 = (ircom - ircom1*0x64)/0xA;
   IRSIM2 = ircom2 + 0x30;
   ircom3 = (ircom - ircom1*0x64 - ircom2*0xA);
   IRSIM3 = ircom3 + 0x30;

   if (IRCOMMAND == 0)
   {
   command_reciev[3] = "#"; // Команда не менялась.
   command_reciev[4] = "f";
   command_reciev[5] = "f";

      CREN =0x0;        // Запрещаем прием.
      RB0 = 0x1;        //Переключим драйвер RS485 на передачу.
      TXEN = 0x1;       // Разрешаем передачу.
      for (i=0; i<6; ++i)   putch(command_reciev[i]);
```

```
        for (i=0;i<1000;i++);     // Задержка для вывода.
        for (i=0; i<6; ++i) command_reciev [i] = " ";
        RB0 = 0x0;        // Выключаем драйвер RS485 на передачу.
        TXEN = 0x0;       // Запрещаем передачу.
        CREN =0x1;        // Разрешаем прием.
    } else    // За время между двумя запросами пришла ИК
команда.
        {
        command_reciev[3] = IRSIM1;
        command_reciev[4] = IRSIM2;
        command_reciev[5] = IRSIM3;

        CREN =0x0;        // Запрещаем прием.
        RB0 = 0x1;        //Переключим драйвер RS485 на передачу.
        TXEN = 0x1;       // Разрешаем передачу.
        for (i=0; i<6; ++i)  putch(command_reciev[i]);
        for (i=0;i<1000;i++);     // Задержка для вывода.
        for (i=0; i<6; ++i) command_reciev [i] = " ";
        RB0 = 0x0;    // Выключаем драйвер RS485 на передачу.
        TXEN = 0x0;  // Запрещаем передачу.
        CREN =0x1;   // Разрешаем прием.
        }
    IRCOMMAND = 0x0;   // Мы передали команду, она больше не
нужна.
}

void main(void)          // Начнем работать.
{
    init_comms();           // Инициализация модуля.
    for (i=0; i<6; ++i) command_reciev [i] = " ";
    command_reciev [0] = "C";
                    //Прочитаем и преобразуем номер модуля.
    MOD_ADDR = PORTB;       // Номер модуля в старших битах.
    MOD_ADDR=MOD_ADDR>>4;      // Сдвинем на четыре бита.
    RA2 = 0x1;            // Чтобы видно, что модуль включен.
// Так и не получилось!!

// Начинаем работать.
start: if (RB3)          // Нет ИК-сигнала.
    {
        PHOTOCOME = 0x0;
        RA0 = 0x00;
```

```
   }
      if (!RB3)    // Появился ИК-сигнал
      {
         for (k=0; k<30;++k);      // Поставим задержку.
         if (!RB3)         // ИК-сигнал не пропал?
         {
            PHOTOCOME = 0x1;
            RA0 = 0x01;
         }
      } else
      {
         PHOTOCOME = 0x0;
         RA0 = 0x00;
      }

while (PHOTOCOME == 1)
{
                // Обработаем ИК-команду.
   ir_cmd ();
break;
}
                // Нет ИК-сигнала, проверим сеть.
      CREN =0x1;
      input = getch();
      switch (input)
      {
      case "C":        // Если обращение к фото модулю.
      for (i=1; i<6; ++i)       // Запишем команду в массив.
      {
         input = getch();
         command_reciev [i] = input;
      }

      MOD_NUM = sim_num_adr();     // Чтение из сети.
      if (MOD_NUM != MOD_ADDR) break;    // Если не наш адрес.
      else
         if (command_reciev [3] = "$") cmd(); // Если
команда.
         default: goto start;
      }
   goto start;
}
```

Схема и программа модуля излучения ИК-кодов

Модуль должен получать команду по сети и излучать ИК-код. В принципе, ИК-коды в «собственном формате» системы могут храниться в EEPROM (можно использовать внешнюю энергонезависимую память). Если мы хотим превратить модуль излучения инфракрасных кодов в универсальный пульт ИК-управления, запоминающий коды других пультов, дополнительная внешняя память может оказаться обязательной составляющей. Для этой цели можно добавить к модулю фотоприемник и подпрограмму считывания–запоминания. Но вернемся к конкретной задаче (рис. 1.53).

Ниже приведена таблица составляющих (табл. 1.7).

Таблица 1.7. Спецификация модуля излучения ИК-кодов

№	Обозначение	Изделие	Количество	Цена (р.)
1	DD1	MAX1483	1	96
2	DD2	PIC16F628A	1	100
3	DD3	LM2936-Z5	1	68
4	VD1	АЛ307	1	3
5	VD2	АЛ144А	1	20
6	C1, C2	0,1 мкФ	2	2
7	C3	100,0 мкФ 16В	1	5
8	R1, R2	1 кОм 0,25 Вт	2	1
9	R3–R6	10 кОм 0,25 Вт	4	1

Ориентировочная стоимость элементов – 297 руб.

Команды:

Ixx$nP и байты данных, где n – количество повторов.

Байты данных в файле input.irc на диске компьютера, записанные с помощью HEX-редактора:

```
02 02 32 B1 26 5D 26 26 26 5D 26 26 26 5D 26 26 26 26 26 5D
26 5D 26 26 26 5D 26 26 FF
```

Рис. 1.53. Принципиальная схема модуля излучения ИК-кодов

Программа модуля излучения ИК-кодов на языке С

Файл заголовка

```
#define MODULNAMESIM "I"

unsigned char getch(void);
int init_comms();
int cmd ();
void ir_trns();
```

Основной файл

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_trans_fnl.h"

unsigned char input;     // Считываем содержимое приемного
регистра.
unsigned char MOD_SIM1;      // Первый символ адреса модуля.
unsigned char MOD_SIM2;      // Второй символ адреса модуля.
unsigned char command_reciev [6]; // Массив для полученной
команды.
```

```
int MOD_ADDR;             // Заданный адрес модуля, как число.
int MOD_ADDR;             // Заданный адрес модуля, как число.
int sim_end_num = 0;
int MOD_NUM;            // Полученный адрес модуля, как число.
int i = 0;
unsigned char b = 0;
int c = 0;
int d = 0;
int k = 0;
int l = 0;
int m = 0;

unsigned char ir_cmd [60];      // Массив для прочитанной ИК-
команды.

unsigned char getch()
{
   while(!RCIF)    // Устанавливается, когда регистр не пуст.
      continue;
   return RCREG;
}

int init_comms()        // Инициализация модуля.
{
PORTA = 0x0;            // Настройка портов А и В.
CMCON = 0x7;
TRISA = 0x0;
TRISB = 0xFE;
RCSTA = 0b10010000;          // Настройка приемника.
TXSTA = 0b00000110;          // Настройка передатчика.
SPBRG = 0x68;             // Настройка режима приема-передачи.
RB0 = 0;            // Выключаем драйвер RS485 на передачу.
CREN =1;
                  // Определим номер модуля.
MOD_ADDR = PORTB;         // Номер модуля в старших битах.
MOD_ADDR=MOD_ADDR>>4;        // Сдвинем на четыре бита.
for (i=0; i<61; ++i) ir_cmd[i] = 0x00;
i = 0;
RA2 = 1;
}
              // Преобразуем символьный адрес в число.
int sim_num_adr()
{
```

```
   sim_end_num = 0;
   MOD_SIM1 = command_reciev [1]; // Первый символ номера.
   MOD_SIM2 = command_reciev [2]; // Второй символ номера.
   MOD_SIM1 = MOD_SIM1 - 0x30;
   MOD_SIM2 = MOD_SIM2 - 0x30;
   sim_end_num = MOD_SIM1*0x0A + MOD_SIM2;
   return sim_end_num;
}

int cmd()
{
      i=0;
                           // Принимаем данные.
      while ((input = getch()) != 0xFF)
      {
      ir_cmd[i] = input;
      ++i;
      }
      ir_cmd[i] = input;
}

void ir_trns()
{
   l = 0;
   for (l = 0; l<4; ++l)
   {
   c = 3;
   while (ir_cmd[c] != 0xFF)       // Наш байт окончания
команды.
     {
                        // Импульс.
      b = ir_cmd[c];
      ++c;

      while (b != 0)
      {
         RA1 = 1;
         RA1 = 1;
         RA1 = 1;
         RA1 = 1;
         RA1 = 1;
         RA1 = 1;
         RA1 = 1;
```

```
            RA1 = 1;
            RA1 = 1;
            RA1 = 1;
            RA1 = 1;
            RA1 = 1;
            RA1 = 0;
            RA1 = 0;
        --b;
        }
                        // Пауза.
        b = ir_cmd[c];
        ++c;

        while (b != 0)
        {
            RA1 = 0;
            RA1 = 0;
            RA1 = 0;
            RA1 = 0;
            RA1 = 0;
            RA1 = 0;
            RA1 = 0;
            RA1 = 0;
            RA1 = 0;
            RA1 = 0;
            RA1 = 0;
            RA1 = 0;
            RA1 = 0;
            RA1 = 0;
        --b;
        }
}
for (m=0; m<1563; ++m);      // Пауза в 25 мс.
}
for (i=0; i<61; ++i) ir_cmd[i] = 0x00;
    i = 0;
}

void main(void)              // Начнем работать.
{
    init_comms();                // Инициализация модуля.
    for (k=0; k<6; ++k) command_reciev [k] = " ";
    command_reciev [0] = "I";
```

```
              //Прочитаем и преобразуем номер модуля.
   MOD_ADDR = PORTB;         // Номер модуля в старших битах.
   MOD_ADDR=MOD_ADDR>>4;        // Сдвинем на четыре бита.

              // Ждем прихода команды и данных.
start: RA2 =1;
      input = getch();
      switch (input)
   {
      case "I":    // Если обращение к модулю ИК команд.
      for (k=1; k<6; ++k)       // Запишем команду в массив.
      {
         input = getch();
         command_reciev [k] = input;
      }

      MOD_NUM = sim_num_adr(); // Определим, к кому адресуются.
      if (MOD_NUM != MOD_ADDR) break;    // Если не наш адрес.
      else
         if (command_reciev [5] == "P")
         {
         cmd();             // Если команда.
         ir_trns();   // Отправим ИК команду на излучатель.
         }
         default: goto start;
      }
   goto start;
}
```

HEX-файл для загрузки в программатор

```
:100000008301003O8A0004282030840078300D20DD
:100010008301 2D2B04068001840A0406031D0A288F
```

```
:1005E2007D0AB500F9003408F8000800830185018E
:1005F20007309F0083168501FE30860090308312FB
:10060200980006308316980068309900831206100D
:1006120018160608A400A5010430F800250DA50C43
:10062200A40CF80B0F2BAC01AD012D08803AF80099
:10063200803078023D3003192C020318292B2C0834
:100642003C3E840083138001AC0A0319AD0A162BC9
:10065200AC01AD0105150800F722AE01AF013A2B3E
:100662002E08363E8400831320308000AE0A031920
:10067200AF0A2F08803AF80080307802063003195A
:100682002E02031C312B4930B6000608A400A50136
:100692000430F800250DA50CA40CF80B4B2B7C2B79
:1006A200AE01AE0AAF012F08803AF800803078021E
:1006B200063003192E0203186A2BA922A3002E0862
:1006C200363E8400831323088000AE0A0319AF0A62
:1006D200542BDB227808A6007908A7002506031D03
:1006E200742B24082606031D7C2B3B08503A031D5D
:1006F2007C2BC32283230515A922A300493A03199F
:10070200512B7C2B8301B001B101B001B101DE2B71
:100712000330A800A901C52B28083C3E840083139E
:100722000008A200A80A0319A90AA2080319A92B02
:1007320085148514851485148514851485148514EF
:10074200851485148514851485108510A203962BB3
:1007520028083C3E840083130008A200A80A03195B
:10076200A90AA2080319C52B8510851085108510CA
:1007720085108510851085108510851085108510CF
:1007820085108510A203B22B28083C3E84008313F7
:10079200000F8D2BB201B3013308803AF800863086
:1007A20078021B30031932020318DB2BB20A031939
:1007B200B30ACD2BB00A0319B10A3108803AF80006
:1007C20080307802043003193002031C892BAC01FB
:1007D200AD012D08803AF800803078023D300319CF
:1007E2002C020318FD2B2C083C3E8400831380014D
:0E07F200AC0A0319AD0AEA2BAC01AD010800F8
:00000001FF
```

Возвращаясь к представлению в некотором «собственном формате», попробуем «прикинуть», как все будет выглядеть.

EEPROM хранит 128 байт. Если мы будем использовать модуль излучателя ИК-кодов для одного устройства, сколько управляющих команд нужно для него в системе? Возьмем, например, DVD-проигрыватель. Какие команды нужны пользователю, чтобы смотреть фильмы? **Проиграть**, **Стоп**,

Предыдущий фрагмент, **Следующий фрагмент**, **Громче**, **Тише**, **Выключить звук**, команды курсора **Влево**, **Вправо**, **Вверх**, **Вниз**, **Ввод**, **Меню**.

Для ровного счета возьмем 15 команд. Таким образом, на одну команду мы можем потратить 8 байт. Много это или мало? Ответ на вопрос следовало бы искать в следующем разделе – «Модуль считывания инфракрасных кодов», проведя считывание всех интересующих нас ИК-команд.

В данный момент я решаю, что нет смысла хранить коды в модуле воспроизведения ИК-команд, будем хранить их на компьютере. Предварительно примем формат команды вида, показанного в табл. 1.8.

Таблица 1.8. Предполагаемый формат ИК-кода

Устройство	Команда	Повторов	Коэффициент	Частота
? байт	? байт	байт	байт	2 байта
Импульс	**Пауза и т.д.**	**EOF**		
2 байт	2 байта	байт 0		

При этом в отличие от предыдущих модулей, модуль излучения должен получать по сети не только команду, но и данные, следующие за ней. Если мы определим префикс модуля с помощью латинской буквы «I», команда может выглядеть следующим образом: `Ixx$nP` и байты данных, где `n` – количество повторов. Я думаю, что повторов от 0 до 9 должно хватить.

Байты данных начнутся с байта коэффициента и закончатся нулевым байтом. Мы не знаем, сколько именно будет передано байт. Для определенности возьмем 50 байт. Значит, все данные будут переданы приблизительно через 50 мс. После приема данных модуль может начать воспроизведение ИК-кода. Таким образом, задержка между нажатием на клавишу управления и отправкой управляющего ИК-кода будет не более 0,1–0,2 с. Думаю, не слишком много.

Осталось два момента, которые мене сейчас не ясны. Первый – хватит ли места в регистрах контроллера для хранения команды. В первых двух банках памяти 176 байт отведено для регистров общего назначения. Полагаю, этого хватит, но хранение в двух банках...

Второй момент, смущающий меня, относится к некоторой вероятности того, что передаваемые данные, а это байты, совпадут с видом команды. Не думаю, что это очень вероятно, но эту неприятность мы можем отслеживать на этапе записи ИК-кода. Если, паче чаяния, это произойдет, воспользуемся коэффициентом пересчета, преобразовав данные к другому виду.

Теперь мы готовы начать разработку нового модуля.

За основу возьмем предыдущий модуль, у которого используем вывод RA6 на выход. К этому выводу порта мы будем подключать светодиод ИК-диапазона (излучатель, или ИК-эмиттер).

Как обычно, я создаю папку в рабочей папке MPLAB, которую называю ir_trans, и копирую в нее все файлы предыдущего модуля. Теперь я открываю в программе MPLAB предыдущий проект, но уже из этой папки. Сохраняю его под новым именем – ir_trans, а также файл заголовков и программы, меняю их в менеджере проекта, включая файл todo.txt, настраиваю установки Debugger. Осталось поменять файлы сценария, удалить из папки все старые файлы, и можно начинать работу.

Вся эта процедура необязательна. Можно открыть новый проект, но при этом не следует забывать, что все его настройки следует сделать заново. Задать рабочую тактовую частоту контроллера и скорость выполнения анимации. Иначе можно наступить на грабли, которые подстерегали нас в самом начале работы.

В отличие от предыдущих данный модуль при настройке должен прочитывать из файла input.txt последовательность данных. Вернемся к рекомендациям изготовителя микроконтроллера PIC16F628A и после строки команды в файле input.txt впишем данные в виде шестнадцатеричных чисел, записанных в столбик и завершаемых командой 0Ah. Но сначала подготовим их. Возьмем код из раздела «Модуль считывания инфракрасных кодов».

```
pulse 2455 Заголовок
space 532
pulse 1291 Единица
space 506
```

```
pulse 664  Ноль
space 533
```
pulse 1266 Единица
space 533
```
pulse 665  Ноль
space 591
```
pulse 1211 Единица
space 533
```
pulse 685  Ноль
space 515
pulse 693  Ноль
space 526
```
pulse 1271 Единица
space 506
pulse 1265 Единица
space 533
```
pulse 666  Ноль
space 557
```
pulse 1241 Единица
space 533
```
pulse 664
```

input.txt

```
"R03$0N"
"I03$5P"  Команда обращения к модулю трансляции ИК команды
01     Количество повторов
01     Коэффициент
25     Частота несущей 37 кГц
97     Младший байт импульса заголовка (шестнадцатеричного
числа)
09     Старший байт импульса заголовка
14     Младший байт паузы
02     Старший байт паузы
E3     Младший байт импульса единицы (1251 мкс - среднее)
04     Старший байт импульса единицы
4A     Младший байт паузы единицы (586 мкс - среднее)
02     Старший байт паузы единицы
4A     Младший байт импульса нуля (586 мкс - среднее)
02     Старший байт импульса нуля
4A     Младший байт паузы нуля (586 мкс - среднее)
02     Старший байт паузы нуля
E3     Младший байт импульса единицы (1251 мкс - среднее)
04     Старший байт импульса единицы
4A     Младший байт паузы единицы (586 мкс - среднее)
```

```
02    Старший байт паузы единицы
4A    Младший байт импульса нуля (586 мкс - среднее)
02    Старший байт импульса нуля
4A    Младший байт паузы нуля (586 мкс - среднее)
02    Старший байт паузы нуля
E3    Младший байт импульса единицы (1251 мкс - среднее)
04    Старший байт импульса единицы
4A    Младший байт паузы единицы (586 мкс - среднее)
02    Старший байт паузы единицы
4A    Младший байт импульса нуля (586 мкс - среднее)
02    Старший байт импульса нуля
4A    Младший байт паузы нуля (586 мкс - среднее)
02    Старший байт паузы нуля
4A    Младший байт импульса нуля (586 мкс - среднее)
02    Старший байт импульса нуля
4A    Младший байт паузы нуля (586 мкс - среднее)
02    Старший байт паузы нуля
E3    Младший байт импульса единицы (1251 мкс - среднее)
04    Старший байт импульса единицы
4A    Младший байт паузы единицы (586 мкс - среднее)
02    Старший байт паузы единицы
E3    Младший байт импульса единицы (1251 мкс - среднее)
04    Старший байт импульса единицы
4A    Младший байт паузы единицы (586 мкс - среднее)
02    Старший байт паузы единицы
4A    Младший байт импульса нуля (586 мкс - среднее)
02    Старший байт импульса нуля
4A    Младший байт паузы нуля (586 мкс - среднее)
02    Старший байт паузы нуля
E3    Младший байт импульса единицы (1251 мкс - среднее)
04    Старший байт импульса единицы
4A    Младший байт паузы единицы (586 мкс - среднее)
02    Старший байт паузы единицы
4A    Младший байт импульса нуля (586 мкс - среднее)
02    Старший байт импульса нуля
00    Завершающий нулевой байт
```

Конечно, файл не должен содержать моих пометок. Я использовал одинаковые усредненные числа. Прав ли я? Это можно будет проверить только при воспроизведении. На данном этапе это не играет роли. Позже, возможно, мы напишем программу, которая будет упаковывать считанные данные. Посмотрим.

Теперь нам нужно позаботиться о том, где мы будем хранить полученные данные. Для этой цели используем массив беззнаковых символов:

```
unsigned char ir_cmd [53];
```

Я задал массив длинною в 53 байта. Вообще мы не знаем, какой длины получится массив, но можно вернуться к этому позже.

Основной переделке подвергается функция прочитывания команды. За последним символом команды 'P' следуют данные. Количество повторов мы получаем перед командой. Но мы оставили под это байт данных. Поэтому пока будем игнорировать количество повторов в команде.

Ожидаем 'P', начинаем заполнение массива.

Основная часть программы:

```
                 // Начинаем работать.
                 // Ждем прихода команды и данных.
start: while(input != MODULNAMESIM) input = getch();
// Ждем обращения к модулю.
      cmd ();              // Обработаем сетевую команду.
      goto start;

// Обработка сетевой команды.
int cmd()
{
   MOD_NUM = sim_num_adr();     // Чтение из сети (файла).
   if (MOD_NUM == MOD_ADDR)     // Если наш адрес модуля.
   {
   while (input != "P")  input = getch();    // Ждем.
      if (input == "P")  // Если символ завершения команды.
      {
                 // Принимаем данные.
      while (i<54)
         {
input = getch();
         ir_cmd[i] = input;
         ++i;
         }
      }
   }
}
```

Перед запуском программы я не забываю установить в высокое состояние биты, имитирующие переключатель адреса модуля (RB4, RB5 в окне Stimulus Controller).

Очередные грабли, на которые я наступаю, – программа не работает. После первого символа в регистре приемника USART появляется 02. Это не номер модуля, дальше ничего не получается. Не пытаюсь разобраться, просто добавляю в файл input.txt еще пару команд перед командой обращения к модулю.

```
"R02$1N"
"R01$1N"
"I03$5P"
```

Теперь программа работает, массив заполняется. Добавив его в наблюдение, можно посмотреть, как он заполняется, и есть ли печатные символы в наших данных.

После заполнения массива мы готовы передать ИК-команду.

У нас три временных интервала:

```
pulse 2455 Заголовок
space 532
pulse 1291 Единица
```

Несущая частота – 37 кГц период ~27 мс. Через время ~13 мс будем менять состояние вывода RA6 порта А, если у нас в массиве записан импульс.

Можно, как в программе предыдущего модуля, использовать встроенные таймеры контроллера. Но для начала попробуем использовать простые циклы.

```
void ir_trns()
{
   i = 0;
   c = 3;       //  В первых трех байтах служебная информация.
   while (ir_cmd[i] != 0)   // Наш байт окончания команды.
   {
               // Импульс.
   b = ir_cmd[c+1];        //  Сначала прочитаем старший байт.
   b = b<<8;        // Сместимся на один байт.
   b = b + ir_cmd[c]; //Сложим байты, получая длительность
импульса.
   c = c + 2;          // Сместимся на два байта.
```

```
   while (b != 0)
   {
      for (d=0;d<14;++d) RA6 = 1; // Теперь с частотой в
37 кГц
      for (d=0;d<14;++d) RA6 = 0; // будем создавать несущую
// частоту.
      --b;
   }
                    // Пауза.
   b = ir_cmd[c+1];
   b = b<<8;
   b = b + ir_cmd[c]; //В этой переменной мы храним
длительность паузы.
   c = c + 2;
   while (b != 0)
      {
      for (d=0;d<28;++d) RA6 = 0; // Передаем паузу.
      --b;
      }
   }
}
```

В глобальные переменные я добавил несколько индексных переменных. Их можно добавить локально, но глобальные переменные удобнее наблюдать.

```
int i = 0;
int b = 0;
int c = 0;
int d = 0;
```

Пытаясь передать массив, я обнаружил, что по умолчанию он не обнуляется, поэтому в раздел инициализации модуля я добавляю обнуление массива. Проблема, которая возникает без этого обнуления, состоит в том, что мы записываем байт, а работаем с двухбайтовыми числами. Нулевой байт записывается в конец массива, но второй байт не нулевой, тогда как условие

```
while (b != 0)        // где b - целое.
for (i=0; i<61; ++i) ir_cmd[i] = 0;  // Обнуление массива.
```

теперь, вроде бы все работает, но мне хочется посмотреть, что транслируется излучателем, соединенным с выводом

RA6. Программа позволяет воспользоваться программным логическим анализатором. Для этого я выделяю всю часть `void ir_trns()`, правой кнопкой мыши вызываю раскрывающееся меню, в котором выбираю раздел **Add Filter ➡ in Trace** (Добавить фильтр ➡ в Трассировку). Теперь, во **View** (Вид) основного меню добавляю к рабочему окну **Simulator Trace** (Симулятор трассировки) и жду, когда заполненный массив начнет транслироваться.

Я ожидаю, что мне удастся увидеть весь передаваемый через RA6 код. Запускаю **Simulator Logic Analyzer** (Симулятор логического анализатора), в котором с помощью кнопки **Channels** (Каналы) вхожу в меню выбора отслеживаемых выводов. Затем выбираю RA6 и добавляю вывод в наблюдение – **Add** (Добавить).

Вопреки желаемому, потратив достаточно много времени, получаю очередное напоминание о «граблях». Удается зафиксировать только фрагменты. Пытаюсь добавить управление еще одним выводом порта A – RA0, устанавливая единицу перед формированием импульса и сбрасывая в ноль при паузе. Единственное, что получилось, – это ряд фрагментов, показанных на рис. 1.54–1.56.

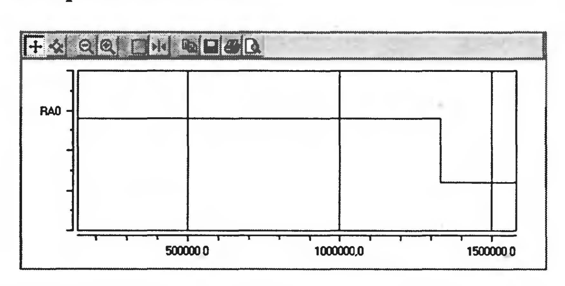


Рис. 1.54. Первый фрагмент эксперимента с логическим анализатором

Первый из фрагментов относится к моменту завершения передачи заголовка. Во втором следом за заголовком передается единица. На последнем фрагменте видно, что передача несущей сменяется паузой. Не густо.

Оставлю программу модуля, удалив управление выводом RA0, и проверю работу модуля, когда соберу его. У меня нет уверенности, что частота несущей – 37 кГц (или близка к ней),

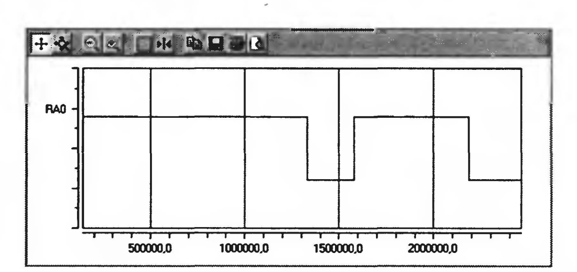


Рис. 1.55. Второй фрагмент эксперимента с логическим анализатором

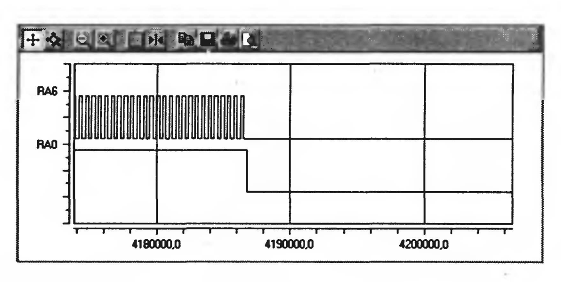


Рис. 1.56. Третий фрагмент эксперимента с логическим анализатором

а длительность импульсов и пауз не требует дополнительной калибровки. Пока я не могу сделать больше, оставлю все, как есть.

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_trans.h"

unsigned char input;      // Считываем содержимое приемного
регистра.
unsigned char MOD_SIM1;      // Первый символ адреса модуля.
unsigned char MOD_SIM2;      // Второй символ адреса модуля.
int MOD_ADDR;             // Заданный адрес модуля, как число.
int MOD_ADDR;             // Заданный адрес модуля, как число.
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
int MOD_NUM;           // Полученный адрес модуля, как число.
int i = 0;
int b = 0;
int c = 0;
```

```
int d = 0;
unsigned char ir_cmd [60];

unsigned char getch()
{
   while(!RCIF)    // Устанавливается, когда регистр не пуст.
      continue;
   return RCREG;
}
                 // Вывод одного байта.
void putch(unsigned char byte)
{
   PORTB = 1;          //Переключим драйвер RS485 на передачу.
   TXEN = 1;       // Разрешаем передачу.
   while(!TXIF)        // Устанавливается, когда регистр пуст.
      continue;
   TXREG = byte;
}

int init_comms()        // Инициализация модуля.
{
PORTA = 0x0;            // Настройка портов А и В.
CMCON = 0x7;
TRISA = 0x80;
TRISB = 0xF6;
RCSTA = 0x90;           // Настройка приемника.
TXSTA = 0x4;       // Настройка передатчика.
SPBRG = 0x16;           // Настройка режима приема-передачи.
INTCON=0;          // Запретить прерывания.
PORTB = 0;              // Выключим передатчик драйвера RS485.
              // Определим номер модуля.
MOD_ADDR = PORTB;       // Номер модуля в старших битах.
MOD_ADDR=MOD_ADDR>>4;     // Сдвинем на четыре бита.
for (i=0; i<61; ++i) ir_cmd[i] = 0;
}
              // Преобразуем символьный адрес в число.
int sim_num_adr()
{
      sim_end_num = 0;
      sim1_num = getch();      //Чтение первого символа
номера.
      MOD_SIM1 = sim1_num;     // Сохраним первый символ.
      sim2_num = getch();      //Чтение второго символа номера.
```

```
      MOD_SIM2 = sim2_num;      // Сохраним второй символ.
      sim1_num = sim1_num - 0x30;
      sim2_num = sim2_num - 0x30;
      sim_end_num = sim1_num*0x0A + sim2_num;
      return sim_end_num;
}

int cmd()
{
      MOD_NUM = sim_num_adr(); // Чтение из сети (файла).
      if (MOD_NUM == MOD_ADDR) // Если наш адрес модуля.
      {
      while (input != "P")  input = getch();   // Ждем.
         if (input == "P") // Если символ завершения команды.
         {
            i=0;
               // Принимаем данные.
         while (i<54)
            {  input = getch();
               ir_cmd[i] = input;
               ++i;
            }
         }
      }
}

void ir_trns()
{
   i = 0;
   c = 3;
   while (ir_cmd[i] != 0)   // Наш байт окончания команды.
   {
               // Импульс.
   b = ir_cmd[c+1];
   b = b<<8;
   b = b + ir_cmd[c];
   c = c + 2;
   RA0 = 1;
   while (b != 0)
   {
      for (d=0;d<14;++d) RA6 = 1;
      for (d=0;d<14;++d) RA6 = 0;
      --b;
```

```
   }
                  // Пауза.
   b = ir_cmd[c+1];
   b = b<<8;
   b = b + ir_cmd[c];
   c = c + 2;
   RA0 = 0;
   while (b != 0)
   {
   for (d=0;d<28;++d) RA6 = 0;
   --b;
   }
   }
}

void main(void)        // Начнем работать.
{
   init_comms();           // Инициализация модуля.
                  //Прочитаем и преобразуем номер модуля.
   MOD_ADDR = PORTB;     // Номер модуля в старших битах.
   MOD_ADDR=MOD_ADDR>>4;     // Сдвинем на четыре бита.

                  // Начинаем работать.
                  // Ждем прихода команды и данных.
start: while(input != MODULNAMESIM) input = getch();
// Ждем обращения к модулю.
      cmd ();      // Обработаем сетевую команду.
      ir_trns();      // Отправим ИК команду на излучатель.
      goto start;
}
```

Все получается разумно и красиво, логично и грамотно. Ай, да я!

Смущает одно. Когда я проверяю времена, соответствующие картинкам анализатора, получаю совсем не то, что хотелось бы. В моем приборном арсенале нет записывающего осциллографа. Представив, как я пытаюсь поймать и измерить последовательность импульсов ИК-команды обычным осциллографом (если вам не приходилось, попробуйте), я решаю вернуться к MPLAB. Верю я тем, кто создал эту замечательную программу! И начинаю понимать, что совсем не «Ай, да я!». Где-то я ошибаюсь.

Напишем простую программку:

```
#include <pic16f62xa.h>
#include <stdio.h>

int d = 0;
int c = 0;

int init_comms()        // Инициализация модуля.
{
PORTA = 0x0;            // Настройка портов A и B.
CMCON = 0x7;
TRISA = 0x80;
TRISB = 0xF6;
RCSTA = 0x90;           // Настройка приемника.
TXSTA = 0x4;            // Настройка передатчика.
SPBRG = 0x16;           // Настройка режима приема-передачи.
INTCON=0;               // Запретить прерывания.
PORTB = 0;              // Выключим передатчик драйвера RS485.
}

void main(void)
{
      init_comms();       // Здесь поставим точку останова.
// Начинаем работать.

start:   for (d=0;d<2;++d) RA6 = 1;
      for (c=0;c<2;++c) RA6 = 0;
      goto start;
}
```

Настроим, установив тактовую частоту контроллера 4 МГц, все, что настраиваем обычно. Выделим с помощью **Add Filter ➡ in Trace** строки, выделенные в тексте, и в пошаговом режиме сделаем четыре шага по программе. Посмотрим, что нам рисует логический анализатор (рис. 1.57).

Время, определяемое в циклах команд контроллера для состояния «0», – около 75 циклов, или 75 мкс. Но я устанавливаю, как я полагал, вывод RA6 в ноль на один цикл. В состоянии «1», что по моим предположениям тоже должно соответствовать 1 мкс, я «зависаю» на 78 мкс. Проверяю маркером (рис. 1.58).

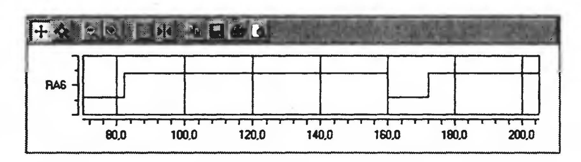


Рис. 1.57. Отображение логическим анализатором пошаговых команд

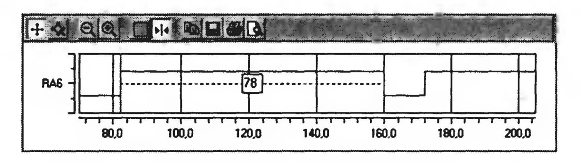


Рис. 1.58. Измерение интервала времени маркером

Так и есть. Не нравится мне это, но пока я не понимаю, в чем дело.

Включив в Simulator Trace режим просмотра в инженерных единицах, я получаю следующую картинку за один проход моих циклов `for` (вид программы в трассировщике MPLAB) – рис. 1.59. Это не полная картина трассировки, строки следуют до номера 45, что соответствует моменту времени в 115 мкс от начала процесса.

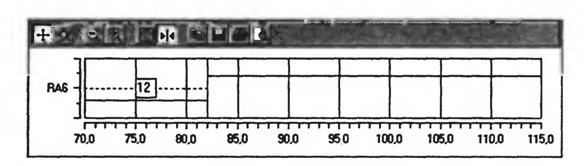


Рис. 1.59. Определение интервала времени одного прохода цикла `for`

И я медленно начинаю понимать...

Это в своем правильном логическом мире, описанном языком C, я за одну команду успеваю перейти от «0» к «1» и обратно. В реальном мире контроллера это занимает ровно столько времени, сколько требует команд. На 82-й микросекунде текущего времени я устанавливаю RA6 в единицу, что трассировщик отображает в 12-й строке командой `BSF 0x5, 0x6` (рис. 1.60).

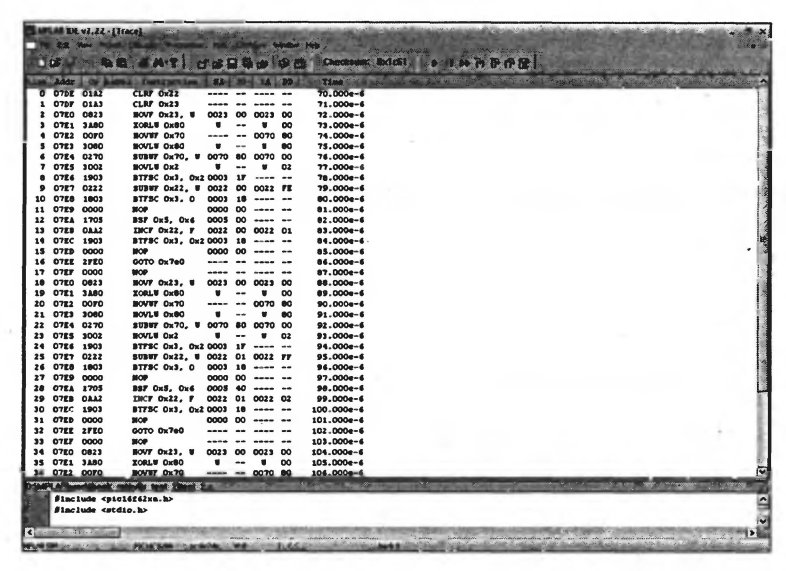


Рис. 1.60. Вид программы в трассировщике MPLAB

Спасибо программистам, создавшим программу MPLAB – сидеть бы мне за осциллографом, чертыхаясь, долго и безуспешно!

Спасибо тем, кто учил меня писать программы в машинных кодах, когда я начал заниматься микропроцессорной техникой; тем, кто терпеливо пояснял мне, как правильно отсчитывать «branch» в командах!

Посыпав голову пеплом, как и полагается, попробуем изменить ситуацию к лучшему. В первую очередь, постараемся добиться, чтобы циклы были одинаковы. Я же хочу получить меандр частотой 37 кГц.

Экран в данный момент выглядит, как показано на рис. 1.61 (я добавил внешний цикл `for`).

Изменив часть программы и манипулируя числами, я несколько меняю результат (рис. 1.62):

```
start: a = 0;
    while (a<100)
```

```
{
for (d=0;d<3;++d) RA6 = 1;
for (c=0;c<2;++c) RA6 = 0;
++a;
}
goto start;
```

Рис. 1.61. Экран отладки в MPLAB

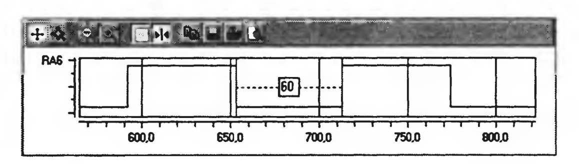


Рис. 1.62. Исправленный интервал времени несущей частоты

Меандр я получаю, но частота несущей остается близка к 8 кГц, тогда как мне нужна частота раз в пять выше. Пока я вижу несколько решений – повысить в пять раз тактовую частоту контроллера, настроить внешний генератор на частоту 37 кГц или использовать внутренние резервы контроллера.

Попытка использовать таймер несколько улучшила ситуацию, но не в полной мере. Последний вариант, не затрагивающий коренных переделок всей программы или модуля, выглядит следующим образом:

```
start: a = 0;
for (a=0; a<100;++a)
    {
       RA6 = 1;
       RA6 = 1;
       RA6 = 1;
       RA6 = 1;
       RA6 = 1;
       RA6 = 1;
       RA6 = 1;
       RA6 = 1;
       RA6 = 1;
       RA6 = 1;
       RA6 = 0;
       RA6 = 0;
    }
    goto start;
```

Получившийся при этом сигнал показан на рис. 1.63.

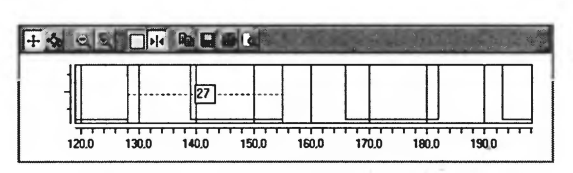


Рис. 1.63. Исправленное значение величины несущей частоты

Период сигнала в 27 мкс соответствует частоте несущей 37 кГц. Это не самое изящное решение. В первую очередь, теперь следует отказаться от изменения несущей частоты ИК-команды «на лету». Модуль будет работать с фиксированной несущей частотой. Посмотрим, можно ли поправить программу модуля.

Не забудьте проверить модуль приема ИК-кодов. Нет ли и там подобной ошибки!

С несущей частотой положение несколько улучшилось, но времена посылок раз в 10 превышают реальные. Первое, что мне приходит в голову, – уменьшить в 10 раз времена в файле input.txt. В этом есть и привлекательная сторона – теперь массив, в котором я храню времена, становится не двухбайтовым, а однобайтовым. Но посылки все еще слишком длинные.

Укорачиваем их еще вдвое, разделив значение времени на два уже в программе. И последнее «заметание мусора под ковер» – деление в файле input.txt всех значений на 1,38.

Мне не нравится то, что я делаю. Но в итоге я получаю результат, показанный на рис. 1.64.

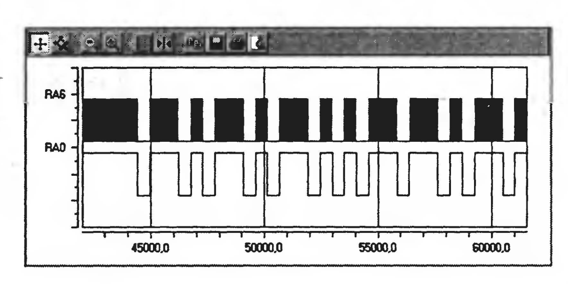


Рис. 1.64. Вид ИК-сигнала на экране логического анализатора

Это даже больше, чем я ожидал. Вернее, я ожидал этого в начале работы, но, не справившись с логическим анализатором, отказался от попыток получить подобную картинку. Здесь несущая, если ее увеличить, выделив затемненный участок сигнала RA6 с помощью мыши и клавиши выбора на инструментальной панели (пятая слева), выглядит, как показано на рис. 1.65, заголовок имеет время, как на рис. 1.66, а единица и пауза, рис. 1.67 и 1.68, соответственно.

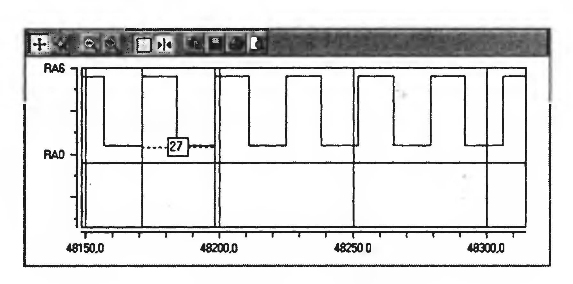


Рис. 1.65. Вид несущей частоты

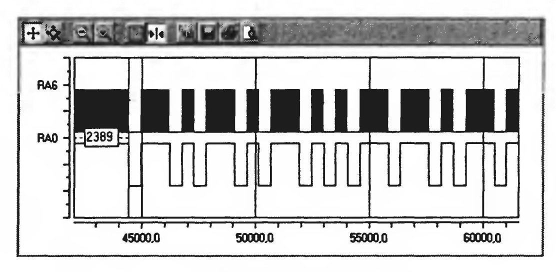


Рис. 1.66. Вид заголовка

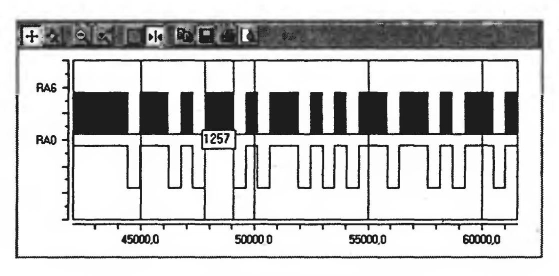


Рис. 1.67. Вид единицы

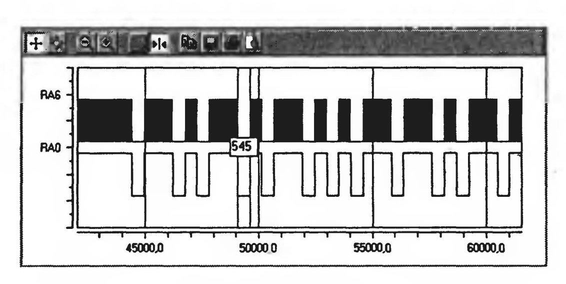


Рис. 1.68. Вид паузы

Времена близки к оригиналу. Последние грабли, на которые я умудряюсь наступить – в функции обработки массива:

```
void ir_trns()
{
   i = 0;
   c = 3;
   while (ir_cmd[i] != 0)   // Наш байт окончания команды.
```

забываю, заменить i на c. Вот так:

```
void ir_trns()
{
   i = 0;
   c = 3;
   while (ir_cmd[c] != 0)   // Наш байт окончания команды.
```

Без этого, закончив передачу ИК-команды, программа меняет байт, из которого я начинаю позже вычитать, и, вычитая из нуля, я получаю опять ненулевой байт, который при проверке циклом while вместо завершения работы функции начинает «пороть чушь». Для отправки я распаковываю массив не с самого начала, а с третьего байта, поскольку в первых трех байтах (от 0 до 2) у меня служебная информация (c = 3), а в этом месте поменять я забыл.

Итоговая программа выглядит так:

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "ir_trans.h"

unsigned char input;     // Считываем содержимое приемного
регистра.
unsigned char MOD_SIM1;      // Первый символ адреса модуля.
unsigned char MOD_SIM2;      // Второй символ адреса модуля.
int MOD_ADDR;          // Заданный адрес модуля, как число.
int MOD_ADDR;          // Заданный адрес модуля, как число.
int sim1_num = 0;
int sim2_num = 0;
int sim_end_num = 0;
int MOD_NUM;        // Полученный адрес модуля, как число.
int i = 0;
int c = 0;
unsigned char ir_cmd [60];
unsigned char b = 0;

unsigned char getch()
{
   while(!RCIF)    // Устанавливается, когда регистр не пуст.
      continue;
   return RCREG;
}
```

```
            // Вывод одного байта.
void putch (unsigned char byte)
{
   PORTB = 1;      //Переключим драйвер RS485 на передачу.
   TXEN = 1;    // Разрешаем передачу.
   while(!TXIF)    // Устанавливается, когда регистр пуст.
      continue;
   TXREG = byte;
}

int init_comms()    // Инициализация модуля.
{
PORTA = 0x0;        // Настройка портов A и B.
CMCON = 0x7;
TRISA = 0x80;
TRISB = 0xF6;
RCSTA = 0x90;          // Настройка приемника.
TXSTA = 0x4;        // Настройка передатчика.
SPBRG = 0x16;          // Настройка режима приема-передачи.
INTCON=0;        // Запретить прерывания.
PORTB = 0;          // Выключим передатчик драйвера RS485.

            // Определим номер модуля.
MOD_ADDR = PORTB;      // Номер модуля в старших битах.
MOD_ADDR=MOD_ADDR>>4;     // Сдвинем на четыре бита.
for (i=0; i<61; ++i) ir_cmd[i] = 0;
}

            // Преобразуем символьный адрес в число.
int sim_num_adr()
{
   sim_end_num = 0;
   sim1_num = getch();       // Чтение первого символа номера.
   MOD_SIM1 = sim1_num;      // Сохраним первый символ.
   sim2_num = getch();       // Чтение второго символа номера.
   MOD_SIM2 = sim2_num;      // Сохраним второй символ.
   sim1_num = sim1_num - 0x30;
   sim2_num = sim2_num - 0x30;
   sim_end_num = sim1_num*0x0A + sim2_num;
   return sim_end_num;
}

int cmd()
```

```
{
      MOD_NUM = sim_num_adr(); // Чтение из сети (файла).
      if (MOD_NUM == MOD_ADDR) // Если наш адрес модуля.
      {
      while (input != "P")  input = getch();   // Ждем.
         if (input == "P") // Если символ завершения команды.
         {
            i=0;
            input = getch();
                       // Принимаем данные.
         while (input != 0x0)
            {
               ir_cmd[i] = input/0x2;
               input = getch();
               ++i;
            }
         }
      }
}

void ir_trns()
{
   i = 0;
   c = 3;
   while (ir_cmd[c] != 0)   // Наш байт окончания команды.
   {
               // Импульс.
   b = ir_cmd[c];
   ++c;

   while (b != 0)
   {
         RA6 = 1;
         RA6 = 1;
         RA6 = 1;
         RA6 = 1;
         RA6 = 1;
         RA6 = 1;
         RA6 = 1;
         RA6 = 1;
         RA6 = 1;
         RA6 = 1;
         RA6 = 1;
```

```
        RA6 = 1;
        RA6 = 1;

        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
            --b;
}
                    // Пауза.
b = ir_cmd[c];
++c;

while (b != 0)
{
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;

        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
        RA6 = 0;
            --b;
}
```

```
   }
}

void main(void)         // Начнем работать.
{
   init_comms();            // Инициализация модуля.
               //Прочитаем и преобразуем номер модуля.
   MOD_ADDR = PORTB;       // Номер модуля в старших битах.
   MOD_ADDR=MOD_ADDR>>4;     // Сдвинем на четыре бита.

               // Ждем прихода команды и данных.
start: while(input != MODULNAMESIM) input = getch(); //Ждем
обращения.

      cmd ();           // Обработаем сетевую команду.
      ir_trns();        // Отправим ИК команду на излучатель.
      goto start;
}
```

Если ее запустить, набраться терпения и дождаться результата, он будет выглядеть, как показано на рис. 1.69 (отслеживание RA0 я убрал, оно было полезно только для определения времен).

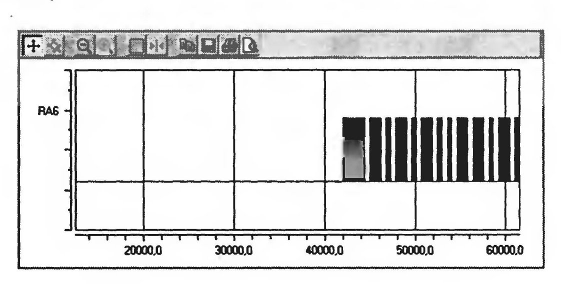


Рис. 1.69. Итоговый вид ИК-кода

Именно такой сигнал мы могли бы увидеть на записывающем осциллографе, если бы воспроизвели ИК-команду с пульта устройства перед фотоприемником (без встроенной обработки сигнала). Картинка не была бы столь четкой, вероятно, но была бы очень похожа. Видно, что сигнал начинается с заголовка (pulse), затем следует короткая пауза (space), после нее – единичная посылка несущей со своей паузой и т.д. По этой картинке можно прочитать сигнал – 10101001101.

Вернувшись к началу этого раздела, мы найдем данную команду.

Проверить работу модуля с этой командой можно несколькими способами (собрав модуль): прочитать команду с пульта и с модуля считывающим устройством и сравнить результаты или отправить команду на устройство, которое этой командой управляется. Работает устройство – работает и команда.

Однако прежде чем проверять работу собранного модуля, есть смысл проверить его работу в MPLAB с другими командами. У меня есть несколько считанных команд, проверкой работы которых я и хочу заняться, записав их для этого в новый input.txt файл. Попробую понять, что из этого получается.

Вот как выглядит команда PLAY одного из устройств фирмы Sharp (в представлении, приведенном к виду, который получился бы с помощью считывания в WinLIRC).

Исходный вид:

```
4000 d101 4000 d101 4000 c700 4000 c700
4000 c700 4000 c700 4000 d101 4000 c700
4000 c700 4000 c700 4000 d101 4000 c700
4000 c700 4000 d101 4000 c700 4000 6f2b
```

После первого преобразования:

```
0040   Импульс
01d1   Пауза
0040   Импульс
01d1   Пауза
0040   Импульс и т.д.
00c7
0040
00c7
0040
00c7
0040
00c7
0040
01d1
0040
00c7
0040
00c7
```

```
0040
00c7
0040
01d1
0040
00c7
0040
00c7
0040
01d1
0040
00c7
0040
2b6f
```

Времена – 64 (0040), 465 (01d1), 199 (00c7), 11119 (2b6f). Первый импульс – 256 мкс. Что предполагает коэффициент 4. Тогда времена – 256 мкс, 1860 мкс, 796 мкс, 44 476 мкс. И, наконец, в том виде, с которым мы начали работать (справа в шестнадцатеричном представлении):

```
pulse 256     12    Я разделил на 13,8 и перевел в HEX-формат
space 1860    87
pulse 256     12
space 1860    87
pulse 256     12
space 796     3A
pulse 256     12
space 796     3A
pulse 256     12
space 796     3A
pulse 256     12
space 796     3A
pulse 256     12
space 1860    87
pulse 256     12
space 796     3A
pulse 256     12
space 796     3A
pulse 256     12
space 796     3A
```

```
pulse 256       12
space 1860      87
pulse 256       12
space 796       3A
pulse 256       12
space 796       3A
pulse 256       12
space 1860      87
pulse 256       12
space 796       3A
pulse 256       12
space 44476     C96
```

Осталось заменить данные в файле input.txt. Посмотрим, что получится.

Общий вид полученных импульсов изображен на рис. 1.70, а времена: первой паузы рис. 1.71, второй паузы и импульса рис. 1.72 и рис. 1.73. И период несущей частоты, рис. 1.74, похож на «настоящий».

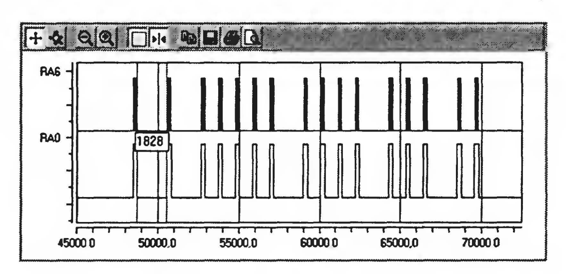


Рис. 1.70. Вид контрольного ИК-кода

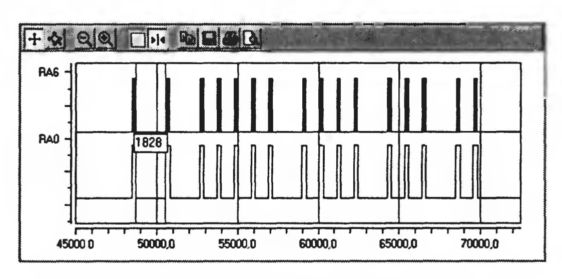


Рис. 1.71. Вид первой паузы контрольного ИК-кода

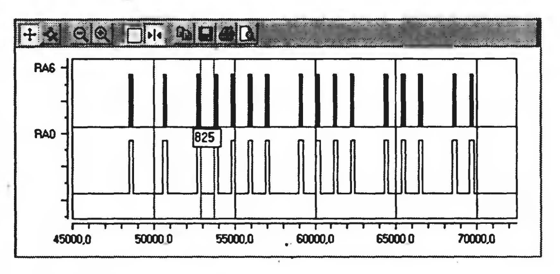


Рис. 1.72. Вид второй паузы контрольного ИК-кода

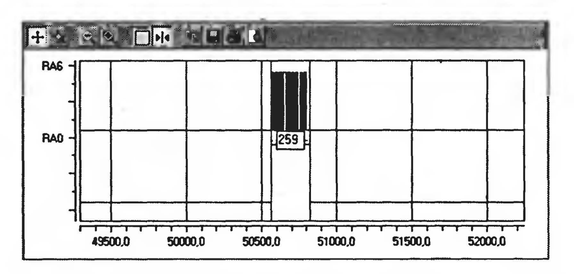


Рис. 1.73. Вид импульса контрольного ИК-кода

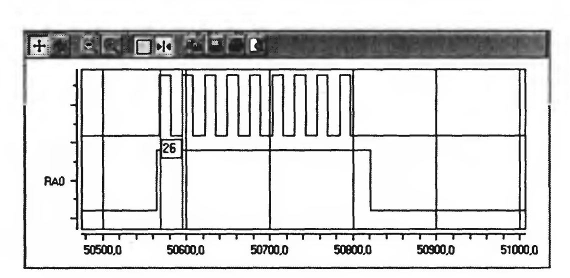


Рис. 1.74. Вид несущей контрольного ИК-кода

Для сравнения посмотрите, как выглядят информационные импульсы в программе, из базы данных которой взяты ИК-коды Sharp (рис. 1.75).

Сравнивая его с видом импульсов, представленных как RA0 на рисунке вида контрольного кода, можно убедиться в их схожести. А времена (256 мкс против 259, 1860 мкс против 1828, 796 мкс против 825) не столь разительно отличаются. Несущая, правда, уехала к 38,5 кГц, но это может быть и не совсем точно измеренное значение (маркер несколько съехал с фронта импульса).

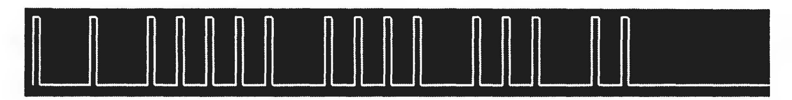


Рис. 1.75. Вид импульсов кода Sharp

И что получилось?

Приступая к разработке, мы вполне разумно определились в том, что хотим от данного модуля. В процессе разработки, столкнувшись с трудностями, мы не менее разумно отказались от части первоначальных запросов. Написав первую версию кода программы, выглядевшую вполне «рассудительно и логично», мы отказались от нее. Предполагая хранить значения текущей длительности интервала в двух байтах, мы для ускорения работы программы приняли однобайтовый вариант хранения.

С точки зрения обучения это яркий пример того, как не надо делать. С практической точки зрения это пример того, как приходится иногда поступать, чтобы справиться с проблемой, обойдя ее хотя бы на время.

В нашем случае, пытаясь овладеть приемами работы с программой MPLAB и одновременно создав некоторую любительскую систему, мы в праве принять несколько решений:

- оставить все, как есть – мы не можем менять частоту несущей ИК-команды «на лету», но пока не убедились, что это нужно, можем не тревожиться; как выглядит программа, изящно или нет, после загрузки ее в контроллер едва ли будет кому-либо интересно; все что требуется, так это чтобы модуль исправно работал;
- можно собрать модуль, опробовать и, если работает, забыть о существовании проблем;
- и, наконец, третье решение –пересмотреть все, что сделано, и создать модернизированные версии тех же модулей; к концу работы появится больше опыта, чаще будет приходить в голову мысль о том, что если бы знать раньше, можно было бы сделать и лучше!

Лично я склоняюсь к тому, что, когда появятся эти мысли, тогда и займемся модернизацией. А пока рисуем схему и отправляемся в «Чип и Дип», собираем модуль.

Напомню формат, который мы «перелопатили», и действия, которые мы осуществляем, вопреки первоначальным замыслам.

Формат записи в файл ИК-команды представлен в табл. 1.9.

Таблица 1.9. Новый вариант формата ИК-кода

Устройство	Повторов	Коэффициент	Импульс	Пауза и т.д.	EOF
байт	байт	байт	байт	байт	байт 0

Импульс и пауза – это времена в мкс, деленные на 13,8 после прочитывания в программе WinLIRC в виде шестнадцатеричных чисел. Затем, в программе, они будут еще раз разделены на 2.

В программе в настоящий момент не используется служебная информация, относящаяся к устройству, количеству повторов и коэффициенту. Позже, если решим модернизировать модуль, мы дополнительно сделаем несколько выходов, которые будут выбираться по записи «Устройство», добавим обработку количества повторов и, возможно, используем коэффициент 13 или 14 для деления времени, полученного при подготовке кода, самой программой. Если решим, что следует внести изменения в модуль.

Модуль считывания ИК-кодов WinLIRC

Будем работать в программе WinLIRC, доступной для свободного использования, с соответствующим модулем в качестве считывающего устройства для предварительной подготовки ИК-кодов. Излучатель WinLIRC, возможно, используем для генерации системных ИК-кодов.

Схема фотосчитывателя и излучателя для работы с программой WinLIRC показана на рис. 1.76. Программа доступна на сайте http://winlirc.sourceforge.net.

Элементы, необходимые для сборки фотосчитывателя приведены в табл. 1.10.

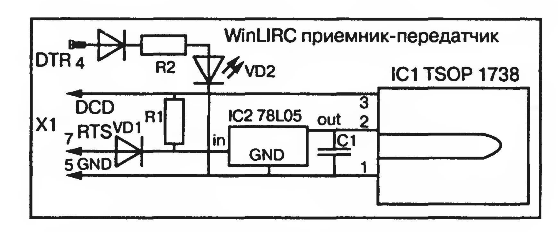


Рис. 1.76. Схема приемника WinLIRC

Таблица 1.10. Спецификация фотосчитывателя

№	Обозначение	Изделие	Кол-во	Цена (р.)
1	IC1	TSOP 1738	1	40
2	IC2	78L05	1	30
3	VD1, VD3	1N4148	2	5
4	VD2	АЛ144А	1	20
5	R1	4,7 кОм 0,25 Вт	1	1
6	R2	2 кОм 0,25 Вт	1	1
7	C1	4,7 мкФ 16 В	1	20
8	X1	DB9 гнездо	1	10

Вид платы может быть таким, как показано на рис. 1.77.

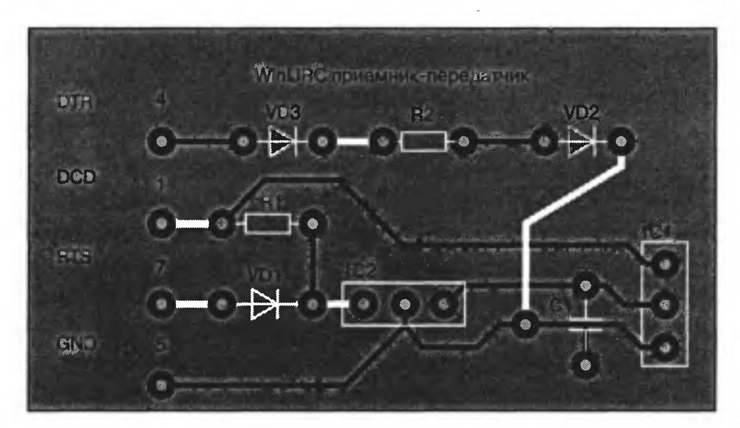


Рис. 1.77. Плата фотоприемника WinLIRC

Программа WinLIRC работает в двух режимах при прочитывании ИК-кодов. В первом режиме она определяет соответствие кода стандарту и создает текстовый файл, в котором записаны параметры кода. Во втором режиме она непосредственно выводит во встроенное окно времена посылок и пауз ИК команды (рис. 1.78).



Рис. 1.78. Вид программы WinLIRC

Появление в правом углу панели значка означает, что программа работает. Правой клавишей мыши с помощью этого значка можно вывести окно программы на экран. Кнопкой **Browse** задать файл конфигурации. А, щелкнув кнопку **Raw Codes**, прочитать ИК-команду.

Думаю, что для считывания команд мы используем второй метод, поскольку существует много кодов, не читаемых первым. Файл, записанный с помощью программы WinLIRC, выглядит следующим образом:

```
Коды с пульта VCR Sony
Прямое считывание, клавиша power
```

```
Outputting raw mode2 data.
space 12856572
pulse 2455
space 532
pulse 1291
space 506
pulse 664
space 533
pulse 1266
space 533
pulse 665
space 591
pulse 1211
space 533
pulse 685
space 515
pulse 693
space 526
pulse 1271
space 506
pulse 1265
space 533
pulse 666
space 557
pulse 1241
space 533
pulse 664
space 24527
pulse 2463
space 579
pulse 1219
space 534
pulse 664
space 534
pulse 1266
space 534
pulse 666
space 531
pulse 1266
space 534
pulse 663
space 536
pulse 662
space 567
```

Реально эта запись получается еще длиннее, если пульт при нажатии клавиши постоянно воспроизводит одну команду до тех пор, пока клавиша не будет отпущена (рис. 1.79).



Рис. 1.79. Считывание ИК-кода программой WinLIRC

Собственно, команда, интересующая нас, содержится в следующей записи:

```
Outputting raw mode2 data.
pulse 2455
space 532
pulse 1291
space 506
pulse 664
space 533
pulse 1266
space 533
pulse 665
space 591
pulse 1211
space 533
```

```
pulse 685
space 515
pulse 693
space 526
pulse 1271
space 506
pulse 1265
space 533
pulse 666
space 557
pulse 1241
space 533
pulse 664
space 24527
```

Здесь `pulse` – вспышка излучателя с частотой, положим, 36 кГц, а `space` – пауза между вспышками.

Раскрасим эту запись в соответствии с тем, что говорилось о кодах Sony выше:

```
pulse 2455   Заголовок
space 532
pulse 1291   Единица
space 506
pulse 664    Ноль
space 533
pulse 1266   Единица
space 533
pulse 665    Ноль
space 591
pulse 1211   Единица
space 533
pulse 685    Ноль
space 515
pulse 693    Ноль
space 526
pulse 1271   Единица
space 506
pulse 1265   Единица
space 533
pulse 666    Ноль
space 557
pulse 1241   Единица
space 533
```

```
pulse 664
space 24527 Пауза между командами
```

Числа здесь – времена в микросекундах. Код, если записать информационное представление, выглядит как 1010100110 1 в двоичном виде или 54D в шестнадцатеричном.

Конечно, хорошо бы хранить команду в информационном представлении, в виде двух байт. В этом случае в EEPROM можно было бы хранить до 64 команд. Но не все ИК-коды таковы. Некоторые имеют до 48 бит, то есть в информационном представлении требуют 6 байт. Есть команды, состоящие из двух последовательно воспроизводимых команд и, вдобавок, длиннее 48 бит.

Можно придумать систему шифрования кодов (с целью их сжатия), но мне кажется, что пока лучше отказаться от идеи хранения кодов в модуле передатчика ИК-команд. До поры до времени будем хранить их на компьютере в файле данных, который будет прочитываться при работе основной программы. При трансляции кода он будет передаваться по системной линии, запоминаться модулем излучения ИК-команд и воспроизводиться им. В этом случае в модуле нам предстоит запомнить только одну команду, которая снимает ограничения на общее количество команд и сложность их хранения в плане длинных кодов.

Итак, мы будем прочитывать ИК-коды с пультов устройств с помощью программы WinLIRC, обрабатывать вручную, записывать их в текстовый файл (или файлы) и воспроизводить по мере необходимости, передавая соответствующему модулю. В этом случае формат команды можно оставить таким, каким мы пользовались при работе с модулем трансляции.

На этом мы можем завершить первую версию в части исполняющих модулей. Пришло время создания основной программы для управляющего компьютера.

Программа для управляющего компьютера

До начала работы еще раз перечислим модули, которые мы разработали для системы:

- релейный модуль;
- модуль приема системных ИК-команд (от старого пульта);
- модуль трансляции ИК-команд для управления бытовой аппаратурой.

Модулей немного. Но даже на их основе можно создать множество очень интересных версий системы, например «Умный кукольный домик» (Барби – достаточно богатая кукла, чтобы иметь свой умный домик) для дочери или младшей сестренки.

Если не усложнять задачу по созданию управляющей программы, в качестве среды разработки основной программы и отладки системы я предлагаю использовать Visual Basic или любую доступную и удобную для вас среду разработки. Я выбрал Visual Basic только по той причине, что «это у меня есть». Вдобавок, вариант Visual Basic упрощает работу с COM-портом. Я пробовал создание подобной среды программирования на языке C++ в KDevelop под управлением операционной системы Linux. Все работает. Все удобно. Среда разработки KDevelop входит в дистрибутив Linux, который стоит около 350 рублей. Тоже, как мне кажется, очень хороший вариант.

Перечислим команды модулей, которые могут потребоваться при написании основной программы.

Релейный модуль. Возможно, мы не будем запрашивать статус реле, используя только команды включения и выключения. Возможно, используем три модуля с адресами 01, 02, 03, которые будут располагаться в трех комнатах и включать настольную лампу в кабинете, торшер в гостиной и бра в холле:

- включить – `Rxx$xN`;
- выключить – `Rxx$xF`, где xx – это 01, 02, 03; x после символа команды «$» – это «1» (будем использовать только по одному реле в модуле);
- передать статус – `Rxx$xS`; при передаче статуса используются следующие символы:
  - включено – `Rxx#xN`;
  - выключено – `Rxx#xF`.

Модуль приема системных ИК-команд. Используем один модуль, который расположим в гостиной.

Запрос статуса – Cxx$0S (аналогично команде запроса статуса релейного модуля).

Ответ модуля – Cxxkkk, если команда пришла, и Cxx#ff, если ИК-команда не приходила.

Модуль трансляции ИК кодов. Предположим, что мы по команде с системного пульта будем включать и выключать телевизор. То есть, нам потребуется отправлять команду power для телевизора, которую предварительно прочитаем с помощью WinLIRC и обработаем, записав в файл с расширением .irc.

Какой я представляю себе в настоящий момент работающую программу? При запуске программы отображается пользовательский интерфейс, который проверяет работу модулей и запускает основную программу. Помещений, где будут установлены модули, три – кабинет, гостиная, холл.

В Visual Basic, как и в других средах программирования, есть удобное средство создания пользовательского интерфейса программы – форма (Form). Этим мы и воспользуемся.

Заменим название формы (свойство Caption) на «Кукольный умный домик». В редакторе изображений создадим иконку, которую вставим, используя свойство **Icon** . Теперь откроем три **Frame**, которые назовем соответственно **Кабинет**, **Гостиная**, **Холл**. Получим картинку, показанную на рис. 1.80.

В редакторе изображений нарисуем изображения выключенной и включенной ламп – и . Их мы используем впоследствии для отображения состояния ламп (или реле). Вставим картинки в каждое из ранее нарисованных помещений на форме (добавив **PictureBox** с вкладки **General**), добавим клавиши, которые будут управлять этими лампами (с помощью реле). Картинки наложим одна поверх другой. Получим вид формы, показанный на рис. 1.81 (последние две картинки не совмещены).

Напомню, что картинки с изображением включенной лампы мы наградим свойством Visible False, а для ввода кода нужно дважды щелкнуть кнопку на форме, например «Лампу включить», и вписать небольшой текст:

```
Private Sub Reley01_on_Click()
Form1.Lamp_off.Visible = False
```

Рис. 1.80. Окно основной программы проекта

Рис. 1.81. Форма программы проекта с расставленными лампами

```
Form1.Lamp_on.Visible = True
End Sub
```

В этом фрагменте я выделил то, что создает Visual Basic при щелчке кнопки на форме.

Расположим в гостиной пульт управления, с которого будем принимать команды (он сыграет роль модуля фотоприемника) и телевизор, которым мы будем управлять с помощью модуля излучателя ИК-команд. Картинки нарисуем в графическом редакторе и наложим одну на другую (конечно, соответствующие, как с лампами). Добавим код, аналогичный вышеупомянутому фрагменту кода, и получим работающую заготовку, которую, в первую очередь, можно использовать для тестирования модулей. Клавиши включения и выключения ламп будут посылать команды модулям и менять картинки в соответствие с состоянием – «Включено» или «Выключено». Аналогично мы поступим и с остальными модулями. А пока можно проверить отображение будущих идей.

Поскольку мы планируем использовать два режима работы – тестовый и основной, – я думаю, есть смысл сделать главное меню с двумя режимами работы. Для этой цели на форме щелкнем правой кнопкой мыши и в открывшемся меню выберем **Редактор Меню**. С его помощью создадим меню с основным пунктом **Режим** и двумя подпунктами: **Проверка** и **Работа**.

При выборе режима **Проверка** (щелкаем этот раздел меню) мы сделаем видимыми все клавиши:

```
Private Sub mnuTest_Click()
Form1.Reley01_off.Enabled = True
Form1.Reley01_off.Visible = True
Form1.Reley02_off.Enabled = True
Form1.Reley02_off.Visible = True
Form1.Reley03_off.Enabled = True
Form1.Reley03_off.Visible = True
Form1.Reley01_on.Enabled = True
Form1.Reley01_on.Visible = True
Form1.Reley02_on.Enabled = True
Form1.Reley02_on.Visible = True
Form1.Reley03_on.Enabled = True
Form1.Reley03_on.Visible = True
Form1.Photo01_on.Enabled = True
Form1.Photo01_on.Visible = True
Form1.IRemitter01_off.Visible = True
```

```
Form1.IRemitter01_off.Enabled = True
Form1.IRemitter01_on.Visible = True
Form1.IRemitter01_on.Enabled = True
Form1.Photo01_off.Enabled = True
Form1.Photo01_off.Visible = True
End Sub
```

При переходе в режим **Работа** мы их уберем «с глаз долой»:

```
Private Sub muWork_Click()
Form1.Reley01_off.Enabled = False
Form1.Reley01_off.Visible = False
Form1.Reley02_off.Enabled = False
Form1.Reley02_off.Visible = False
Form1.Reley03_off.Enabled = False
Form1.Reley03_off.Visible = False
Form1.Reley01_on.Enabled = False
Form1.Reley01_on.Visible = False
Form1.Reley02_on.Enabled = False
Form1.Reley02_on.Visible = False
Form1.Reley03_on.Enabled = False
Form1.Reley03_on.Visible = False
Form1.Photo01_on.Enabled = False
Form1.Photo01_on.Visible = False
Form1.IRemitter01_off.Visible = False
Form1.IRemitter01_off.Enabled = False
Form1.IRemitter01_on.Visible = False
Form1.IRemitter01_on.Enabled = False
Form1.Photo01_off.Enabled = False
Form1.Photo01_off.Visible = False
End Sub
```

Теперь работающая программа будет выглядеть в двух режимах, как показано на рис. 1.82.

В программе я использовал обозначения Reley01_on (off), Photo01_off (on) и IRemitter01_on (off) для клавиш управления **Лампу включить** и т.д. Клавиша **Команда?** – запрос фотоприемнику. Обозначения содержат адрес модулей Reley01, Reley02, Photo01 и т.д.

Не трогая режим работы, попробуем организовать работу с портом COM1. Для этих целей используем возможность добавить элементы управления. Я добавил требуемый элемент

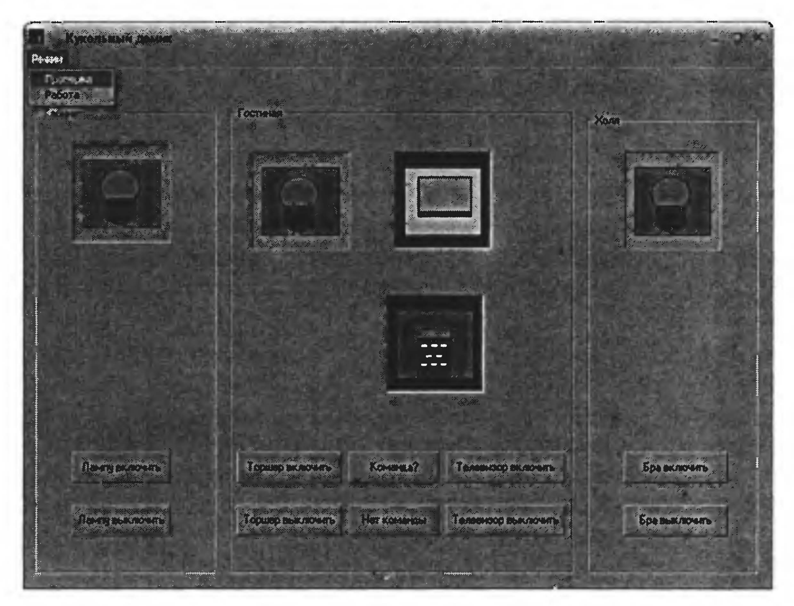


Рис. 1.82. Окно программы проекта с расставленными устройствами и меню

управления **MSComm** на панель **Home** в Visual Basic, которую создал, чтобы не перегружать основную инструментальную панель. Этот элемент переносим на форму для использования в программе.

Для добавления элемента на инструментальную панель открываем в меню **Проект** раздел **Компоненты** и ставим флажок напротив **Microsoft Comm Control 6.0** (рис. 1.83).

После добавления элемента управления на форму он становится доступен для использования (рис. 1.84).

Признаться, я хотел сделать два варианта этой программы: на Visual Basic и Delphi, благо, есть возможность поработать и с той, и с другой средой программирования. Но в Delphi, установив необходимый компонент VCL, я не смог им воспользоваться по причине отсутствия на это лицензии. Может быть, я что-то сделал не так, но не стал без нужды озадачивать тех, кто дал мне возможность поработать на компьютере, и перешел на Visual Basic, решив, что в этой среде Microsoft не

потребует лицензии. Еще раз повторю, что, возможно, что-то сделал не так, как должно в Delphi, но... описывать работу с COM-портом «вручную» в данный момент я считаю преждевременным. Да и зачем, если именно для облегчения работы придуманы VCL, OLE и прочие загадочные составляющие современных сред программирования?

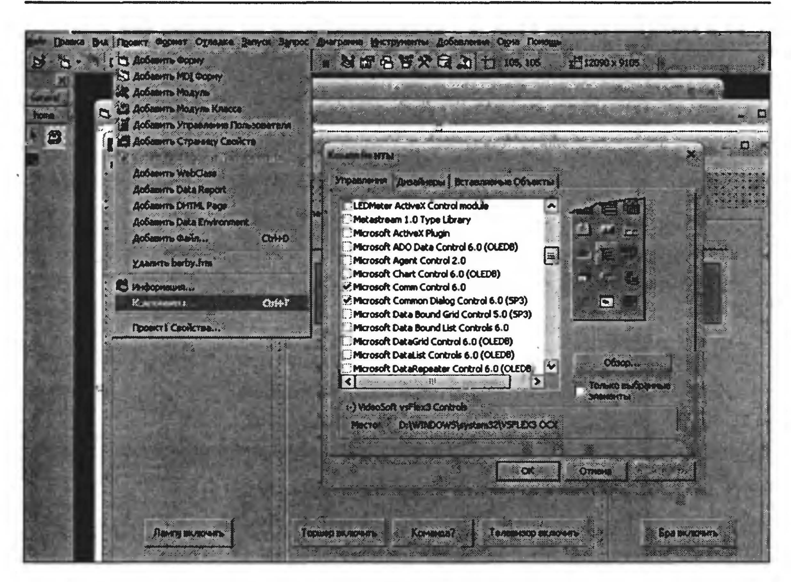


Рис. 1.83. Установка COM-порта в Visual Basic

В режим проверки добавим инициализацию порта:

```
Private Sub mnuTest_Click()
Form1.MSComm1.CommPort = 1
Form1.MSComm1.Settings = "9600,N,8,1"
Form1.MSComm1.PortOpen = True
Form1.Reley01_off.Enabled = True
Form1.Reley01_off.Visible = True
И т.д.
End Sub
```

Как и в рабочий режим. Изменим управление релейными модулями в режиме тестирования, добавив реальные команды.

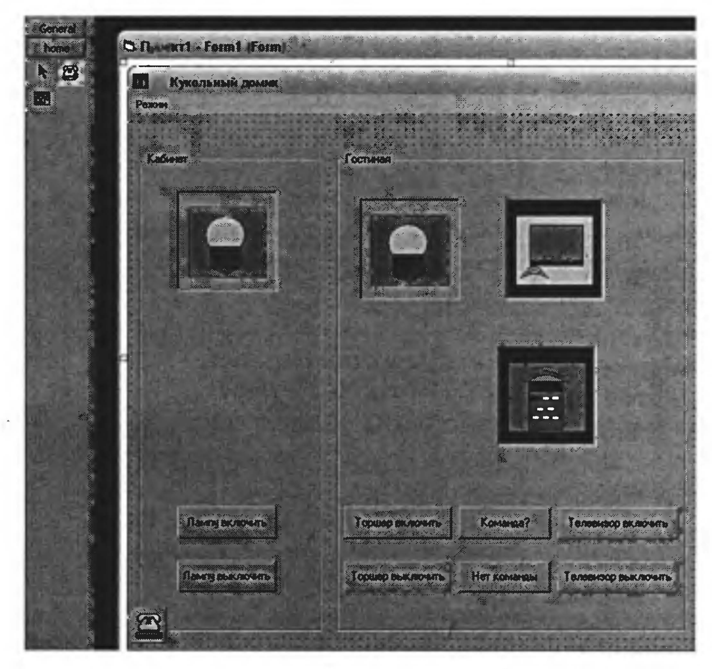


Рис. 1.84. Форма проекта с установленным COM-портом

Команда **Включить лампу**:

```
Private Sub Reley01_on_Click()
Form1.MSComm1.Output = "R01$1N"
Form1.MSComm1.Output = "R01$1S"
If Form1.MSComm1.Input = "R01#1N" Then
Form1.Lamp_off.Visible = False
Form1.Lamp_on.Visible = True
End If
End Sub
```

Команда **Выключить лампу**:

```
Private Sub Reley01_off_Click()
Form1.MSComm1.Output = "R01$1F"
Form1.MSComm1.Output = "R01$1S"
If Form1.MSComm1.Input = "R01#1F" Then
Form1.Lamp_on.Visible = False
```

```
Form1.Lamp_off.Visible = True
End If
End Sub
```

Пока я не вставил код инициализации порта при загрузке программы, поэтому при запуске программы и попытках нажать на клавишу **Включить лампу** появляется сообщение об ошибке. Это сообщение служит напоминанием, что нужно выбрать режим **Проверка**. Вторая особенность работы программы – лампа, которую я хочу включить или выключить, не реагирует на команды. Причина в том, что релейный модуль, настроенный соответствующим образом, я пока не включал.

Будем надеяться, что с включенным модулем программа будет работать правильно. С надеждой на это важное допущение исправим работу всех релейных модулей аналогично первому.

Например, для третьего модуля (реле в модуле №1):

```
Private Sub Reley03_off_Click()
Form1.MSComm1.Output = "R03$1F"
Form1.MSComm1.Output = "R03$1S"
If Form1.MSComm1.Input = "R03#1F" Then
Form1.Bra_on.Visible = False
Form1.Bra_off.Visible = True
End If
End Sub
```

Теперь попробуем обработать прием системной ИК-команды модулем приема ИК-кодов. Для режима проверки нам достаточно одной системной команды. Пусть это будет команда 001.

```
Private Sub Photo01_on_Click()
Form1.MSComm1.Output = "C01$0S"
If Form1.MSComm1.Input = "C01001" Then
Form1.IRcmnd_off.Visible = False
Form1.IRcmnd_on.Visible = True
Else
Form1.IRcmnd_on.Visible = False
Form1.IRcmnd_off.Visible = True
End If
End Sub
```

Работу клавиши **Нет команды** оставим без изменений.

Пришло время разобраться с командой включения телевизора. Подготовим файл, возможно, внеся некоторые исправления в файл input.txt. Мы его использовали с программой MPLAB при разработке модуля ИК-команд. Изменим его расширение, превратив в файл input.irc, который я размещаю на диске D в папке barby:

```
02 02 32 B1 26 5D 26 26 26 5D 26 26 26 5D 26 26 26 26 26 5D
26 5D 26 26 26 5D 26 26 00
```

Я оставил пробелы между числами, чтобы числа легче читались, но эти пробелы не нужны в реальном файле.

```
Private Sub IRemitter01_on_Click()
Form1.TV_off.Visible = False
Form1.TV_on.Visible = True
intFH = FreeFile()
Open "D:\barby\input.irc" For Input As intFH
Do Until EOF(intFH)
Line Input #intFH, strString
strCmnd = strCmnd & strString & vbLf
Loop
Form1.MSComm1.Output = "I01$5P"
Form1.MSComm1.Output = strCmnd
End Sub
```

Вот такая получилась кнопка. Я честно «срисовал» весь фрагмент программы из полного руководства по Visual Basic 6.0 Михаэля Райтигера и Геральда Муча (издание BHV, «Ирина», Киев, 2000), которое обнаружил на книжной полке. Программа не протестует против этого фрагмента, у меня тоже пока нет оснований для протестов.

Вторую клавишу управления телевизором можно снабдить этим же кодом. Многое зависит от команд включения и выключения. Мой старый телевизор включается подачей команды канала, а выключается клавишей **power**. Пока это не принципиально, важно только, чтобы в файле input.irc была правильная команда. Перед проверкой работы программы в режиме тестирования я удалил строки:

```
Form1.MSComm1.PortOpen = False
Form1.MSComm1.CommPort = 1
```

```
Form1.MSComm1.Settings = "9600,N,8,1"
Form1.MSComm1.PortOpen = True
```

из Private Sub muWork_Click(). Обозначил ее свойство **Visible** равным **False** (в редакторе меню убрал флажок в опции **Включено**) и добавил к фрагменту щелчка кнопки меню **Проверка** строку Form1.mnuWork.Enabled = True, а к фрагменту щелчка кнопки меню **Работа** – Form1.mnuTest.Enabled = False. Добавил еще один пункт в основное меню **Стоп**, работу которого описал так:

```
Private Sub mnuStop_Click()
Form1.mnuTest.Enabled = True
Form1.mnuWork.Enabled = False
Form1.MSComm1.PortOpen = False
End Sub
```

Причина этого, полагаю, понятна – «заплатки» для правильной работы COM-порта, который нужно во время открывать и закрывать. Чуть позже поправим (если не забудем) наше лоскутное одеяло, а теперь настал, как говорят, момент истины. До написания блока **Работа** основной программы следует проверить ее тестовую часть. Пришла пора собрать модули, подключить конвертор к компьютеру, а модули – к конвертеру, и запустить программу. Но это завтра. Сейчас время позднее, я не успею дома переписать программу, разложиться возле компьютера и кое-что проверить. Однако меня беспокоит фрагмент чтения из файла. Что же я прочитаю? Попробую в очередной раз схитрить, немного времени у меня еще есть (рис. 1.85).

Хитрость в добавленной кнопке в верхней части с именем wrem, надпись на которой я убрал, а перед отправкой ИК-команды кнопкой **Телевизор включить** добавил строку, выделенную ниже:

```
Private Sub IRemitter01_on_Click()
Form1.TV_off.Visible = False
Form1.TV_on.Visible = True
intFH = FreeFile()
Open "D:\barby\input.irc" For Input As intFH
Do Until EOF(intFH)
Line Input #intFH, strString
```

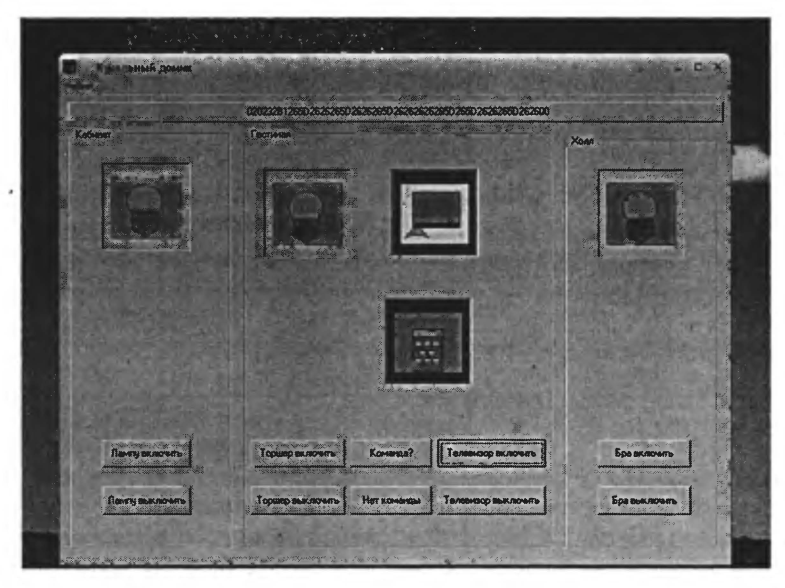


Рис. 1.85. Окно проекта с выводом тестовой информации

```
strCmnd = strCmnd & strString & vbLf
Loop
Form1.wrem.Caption = strCmnd
Form1.MSComm1.Output = "I01$5P"
Form1.MSComm1.Output = strCmnd
End Sub
```

После чтения из файла input.irc и до отправки строки в COM-порт все, что было прочитано, выводится на эту клавишу в качестве названия.

Пока у меня не остается сомнений (спасибо Михаэлю Райтигеру и Геральду Мучу), я могу спокойно провести сегодняшний вечер у телевизора. А завтра начну проверять модули, но это будет завтра...

Завтра

Вот «Завтра» и пришло.

Ранее уже описана проверка релейного модуля и модуля приема системных ИК-команд. Осталось проверить модуль

трансляции ИК-кодов. Предполагаю следующий порядок проверки. Подключаю модуль. Отправляю с компьютера команду трансляции ИК-кода и читаю его с помощью WinLIRC в режиме прямого чтения кода.

Первая правильная мысль – не устанавливать для излучения светодиод ИК-диапазона, а использовать три светодиода АЛ307, которые уже установлены на макетной плате. Возможно, красный светодиод «прихватит» и ИК-диапазон.

Первый результат. После включения клавиши **Телевизор включить** и отправки ИК-кода модуль «виснет», а программа WinLIRC хладнокровно выводит результат:

```
Outputting raw mode2 data.
space 6025710
pulse 1086
space 994
pulse 1074
space 1301
pulse 1057
space 978
pulse 1185
space 1011
pulse 1448
space 979
pulse 1147
space 983
pulse 1155
space 1002
pulse 1158
space 1014
pulse 1441
space 998
pulse 1145
space 992
pulse 1133
space 1002
pulse 1163
space 1012
pulse 1442
space 998
pulse 1136
```

```
space 976
pulse 1157
space 981
pulse 1156
space 977
pulse 1158
space 977
pulse 1130
space 1068
pulse 1467
space 951
pulse 1156
space 1053
pulse 1427
space 978
pulse 1159
space 999
pulse 1160
space 954
pulse 1183
space 1013
pulse 1466
space 950
pulse 1104
```

Я повторяю эксперимент несколько раз, постепенно приходя к выводу, что удобнее фиксировать результат в записи короткие (0) и длинные (1) импульсы, чтобы удобнее было сравнивать с исходным кодом.

Воспроизводимый код: 0001000010000000100000000100000 0000001000100000000100

Повторное воспроизведение дает тот же код, что уже хорошо.

Но оригинальный код имеет вид (в том же виде):

```
Заголовок    01 0001 0001 000001 01 0001 00
```

Структура похожа, но сам код...

Первое, что я делаю, – убираю все отладочные вставки в программе модуля. Попутно «смахиваю» функцию `putch()` – модуль работает «молча» – и убираю настройки контроллера, относящиеся к настройке передатчика. Модуль не подает признаков жизни. Возвращаюсь на исходные позиции. Не помогает.

Изменяю количество байтов, принимаемых в массив записи команды, с 54 до 29 (реальных). И это не помогает.

Возможно, вам быстрее, чем мне, приходит в голову мысль – а что я читаю из файла input.irc? В отладочной программе MPLAB я записывал символы. И читаю, надо понимать, символы. А ожидаю числа.

Открываю файл input.irc в HEX–редакторе и меняю символы на числа. Пробую:

```
Outputting raw mode2 data.
space 10739323
pulse 3650
space 732
pulse 1951
space 733
pulse 834
space 755
pulse 1983
space 684
pulse 859
space 719
pulse 2009
space 667
pulse 833
space 759
pulse 811
space 740
pulse 1937
space 742
pulse 1977
space 704
pulse 844
space 726
pulse 1978
space 705
pulse 832
space 2487801
Заголовок 0100010001000000101000100
```

Пытаюсь разобраться с временами, что не сложно, – уменьшить числа в файле input.irc. Меняю коэффициент с 13,8 на 19. Мне не нравится, что первое нажатие клавиши

Телевизор включить не воспроизводит код, требуется повторное нажатие. Но времена становятся ближе к исходным.

Делаю попытку включить магнитофон. Неудачно. Одна из причин этого может оказаться в единичности посылки кода. «Родные» коды идут сплошным потоком. А если попробовать четыре повтора с паузой в 24505 мс? Пробую переделать программу для реализации четырех повторов. Делаю паузу между повторами с помощью цикла `for`. Заодно пытаюсь понять, отчего и где зависает модуль.

Дополняю еще одно число в файл .irc, чтобы цикл воспроизведения в программе модуля заканчивался правильно. То есть, чтобы программа не «зависала» на отсутствующем числе длительностью в паузу. Попутно меняю условие завершения чтения с количества чисел (29) на байт завершения файла «00».

В какой-то момент видеомагнитофон даже удается включать и выключать, нажимая клавишу **Телевизор включить**. Но модуль продолжает виснуть.

На все это уходит два дня «борьбы с собственной гениальностью». Но есть и моменты проблесков. Проблема, которой закончилась отладка модуля приема ИК-кодов, превратилась в ясное понимание, что лень не всегда хороша.
Чтобы не создавать весь проект полностью, я копирую предыдущий в другую папку, правлю имена файлов и т.д. Но, поправляя программу при отладке, порой забываю, с какой из папок я работаю. Если бы не лень, я создавал бы новый проект, открывал новые файлы... Словом, получилось так, что исходный текст я правил в одной из папок, а компилировал в другую. Даю себе слово впредь в менеджере проекта удалять и добавлять все файлы заново. Посмотрим!

За время отладки в голову приходила мысль, что завершение файла байтом «00», может каким-то образом влиять на зависание. Но как? Когда не остается никаких разумных вариантов, и я устаю от бесконечного перепрограммирования микросхемы, я меняю в файле input.irc завершающий байт с 00 на FF, а коэффициент – на 25.

Модуль перестает «виснуть», магнитофон исправно включается и выключается. Программа модуля имеет к этому времени вид, представленный в начале раздела.

В основной программе фрагменты для клавиш **Телевизор включить** и **Телевизор выключить** к концу наладки выглядят так:

```
Private Sub IRemitter01_off_Click()
intFH = FreeFile()
Open "D:\barby\input_1.irc" For Input As intFH
Do Until EOF(intFH)
Line Input #intFH, strString
Loop
Form1.MSComm1.Output = "I14$5P"
For i = 0 To 10000
Next i
Form1.MSComm1.Output = strString
For i = 0 To 20000000
Next i
Form1.TV_on.Visible = False
Form1.TV_off.Visible = True
End Sub

Private Sub IRemitter01_on_Click()
intFH = FreeFile()
Open "D:\barby\input_1.irc" For Input As intFH
Do Until EOF(intFH)
Line Input #intFH, strString
Rem strCmnd = strCmnd & strString & vbLf
Loop
Rem Form1.wrem.Caption = strCmnd
Form1.MSComm1.Output = "I14$5P"
For i = 0 To 10000
Next i
Form1.MSComm1.Output = strString
For i = 0 To 20000000
Next i
Form1.TV_off.Visible = False
Form1.TV_on.Visible = True
End Sub
```

И немного назад

Поскольку в первой версии единственным средством управления служит старый пульт от видеомагнитофона (или телевизора), хотелось бы быть уверенным, что все будет работать правильно (или, хотя бы, будет работать). Немного переделаю основную программу:

```
Dim ext As Boolean

Private Sub mnuWork_Click()
Form1.mnuTest.Enabled = False
Form1.Command1.Enabled = False
Form1.Command1.Visible = False
Form1.Reley01_off.Enabled = False
Form1.Reley01_off.Visible = False
Form1.Reley02_off.Enabled = False
Form1.Reley02_off.Visible = False
Form1.Reley03_off.Enabled = False
Form1.Reley03_off.Visible = False
Form1.Reley01_on.Enabled = False
Form1.Reley01_on.Visible = False
Form1.Reley02_on.Enabled = False
Form1.Reley02_on.Visible = False
Form1.Reley03_on.Enabled = False
Form1.Reley03_on.Visible = False
Form1.Photo01_on.Enabled = False
Form1.Photo01_on.Visible = False
Form1.Photo01_off.Enabled = False
Form1.Photo01_off.Visible = False
Form1.IRemitter01_off.Visible = False
Form1.IRemitter01_off.Enabled = False
Form1.IRemitter01_on.Visible = False
Form1.IRemitter01_on.Enabled = False
Form1.work
End Sub

Sub work()
Do

Form1.KeyPreview = True
Form1.MSComm1.Output = "C14$0S"
For i = 0 To 10000000
```

```
Next i
strAnsw = Form1.MSComm1.Input
Form1.wrem.Enabled = False
Form1.wrem.Visible = False
Form1.Refresh

Select Case strAnsw

Case "C14$0SC14001"
Form1.Lamp_off.Visible = False
Form1.Lamp_on.Visible = True
Form1.Refresh

Case "C14$0SC14129"
Form1.Lamp_off.Visible = True
Form1.Lamp_on.Visible = False
Form1.Refresh

Case "C14$0SC14193"
Form1.Torsher_off.Visible = False
Form1.Torsher_on.Visible = True
Form1.Refresh

Case "C14$0SC14033"
Form1.Torsher_off.Visible = True
Form1.Torsher_on.Visible = False
Form1.Refresh

Case "C14$0SC14161"
Form1.Bra_off.Visible = False
Form1.Bra_on.Visible = True
Form1.Refresh

Case "C14$0SC14097"
Form1.Bra_off.Visible = True
Form1.Bra_on.Visible = False
Form1.Refresh

Case "C14$0SC14065"
ext = True
End Select

Loop While ext = False
```

```
Form1.Command1.Enabled = True
Form1.Command1.Visible = True
Form1.Refresh
End Sub
```

Переделка касается добавления функции work (), где программа «крутится» в цикле после включения режима **Работа**, ожидая ввода ИК-команд с пульта. Команды включают и выключают «лампы» на форме. Модуль приема системных ИК-команд я проверял и полагал, что с ним не возникнет проблем. Я ошибался!

При тестировании модуля он правильно отвечал на ИК-команды. Но, когда запросы пошли часто, во время прохождения бесконечного цикла основной команды модуль стал «виснуть». Пришлось вернуться к его наладке. В итоге программа модуля приобрела вид, представленный в начале описания модуля.

Теперь модуль не «виснет», основная программа «крутится» в цикле. По командам с пульта лампочки на форме включаются и выключаются, а, нажав на цифру «3» на пульте, я покидаю режим работы и могу выключить основную программу.

Здесь тоже есть момент, который меня не устраивает, но с ним, я думаю, можно разобраться позже. Он относится к выходу из цикла в основной программе. Если я пытаюсь остановить программу с помощью меню или закрыть ее привычным для Windows образом, ничего не получается. Проблема в том, что программа «крутится» в цикле, из которого может выйти только по событиям, включенным в этот цикл. А единственное событие – прием команды, отправляемой клавишей «3» с пульта. Я думаю, это решаемый вопрос, но к его решению мы вернемся позже.

Текст основной программы на языке Visual Basic

```
Dim strAnsw As String
Dim ext As Boolean

Sub work()
Do
Form1.KeyPreview = True
Form1.MSComm1.Output = "D14$0S"
```

```
For i = 0 To 10000000
Next i
strAnsw = Form1.MSComm1.Input
Rem Form1.wrem.Caption = strAnsw
Form1.wrem.Enabled = False
Form1.wrem.Visible = False
Form1.Refresh
Select Case strAnsw
Case "C14$0SC14001"
Form1.Lamp_off.Visible = False
Form1.Lamp_on.Visible = True
Form1.Refresh
Case "C14$0SC14129"
Form1.Lamp_off.Visible = True
Form1.Lamp_on.Visible = False
Form1.Refresh
Case "C14$0SC14193"
Form1.Torsher_off.Visible = False
Form1.Torsher_on.Visible = True
Form1.Refresh
Case "C14$0SC14033"
Form1.Torsher_off.Visible = True
Form1.Torsher_on.Visible = False
Form1.Refresh
Case "C14$0SC14161"
Form1.Bra_off.Visible = False
Form1.Bra_on.Visible = True
Form1.Refresh
Case "C14$0SC14097"
Form1.Bra_off.Visible = True
Form1.Bra_on.Visible = False
Form1.Refresh
Case "D14$0SD14240"
ext = True
End Select
Loop While ext = False

Form1.Command1.Enabled = True
Form1.Command1.Visible = True
Form1.Refresh
End Sub

Private Sub Command1_Click()
```

```
Form1.Command1.Caption = "Test run"
Form1.Command1.Enabled = True
Form1.wrem.Caption = "          "
Form1.MSComm1.Output = "R14$1N"
For i = 0 To 10000000
Next i
strAnsw = Form1.MSComm1.Input
Form1.wrem.Caption = strAnsw
Form1.Command1.Caption = "Test R"
Form1.Command1.Enabled = True
End Sub

Private Sub IRemitter01_off_Click()
intFH = FreeFile()
Open "D:\barby\input_1.irc" For Input As intFH
Do Until EOF(intFH)
Line Input #intFH, strString
Loop
Form1.MSComm1.Output = "I14$5P"
For i = 0 To 10000
Next i
Form1.MSComm1.Output = strString
For i = 0 To 20000000
Next i
Form1.TV_on.Visible = False
Form1.TV_off.Visible = True
End Sub

Private Sub IRemitter01_on_Click()
intFH = FreeFile()
Open "D:\barby\input_1.irc" For Input As intFH
Do Until EOF(intFH)
Line Input #intFH, strString
Rem strCmnd = strCmnd & strString & vbLf
Loop
Rem Form1.wrem.Caption = strCmnd
Form1.MSComm1.Output = "I14$5P"
For i = 0 To 10000
Next i
Form1.MSComm1.Output = strString
For i = 0 To 20000000
Next i
Form1.TV_off.Visible = False
```

```
Form1.TV_on.Visible = True
End Sub

Private Sub mnuStop_Click()
Form1.mnuTest.Enabled = True
Form1.mnuWork.Enabled = False
Form1.MSComm1.PortOpen = False
End Sub

Private Sub mnuTest_Click()
Form1.mnuWork.Enabled = True
Form1.MSComm1.CommPort = 1
Form1.MSComm1.Settings = "2400,N,8,1"
Form1.MSComm1.PortOpen = True
Form1.Reley01_off.Enabled = True
Form1.Reley01_off.Visible = True
Form1.Reley02_off.Enabled = True
Form1.Reley02_off.Visible = True
Form1.Reley03_off.Enabled = True
Form1.Reley03_off.Visible = True
Form1.Reley01_on.Enabled = True
Form1.Reley01_on.Visible = True
Form1.Reley02_on.Enabled = True
Form1.Reley02_on.Visible = True
Form1.Reley03_on.Enabled = True
Form1.Reley03_on.Visible = True
Form1.Photo01_on.Enabled = True
Form1.Photo01_on.Visible = True
Form1.IRemitter01_off.Visible = True
Form1.IRemitter01_off.Enabled = True
Form1.IRemitter01_on.Visible = True
Form1.IRemitter01_on.Enabled = True
Form1.Photo01_off.Enabled = True
Form1.Photo01_off.Visible = True
End Sub

Private Sub mnuWork_Click()
Form1.mnuTest.Enabled = False
Form1.Command1.Enabled = False
Form1.Command1.Visible = False
Form1.Reley01_off.Enabled = False
Form1.Reley01_off.Visible = False
Form1.Reley02_off.Enabled = False
```

```
Form1.Reley02_off.Visible = False
Form1.Reley03_off.Enabled = False
Form1.Reley03_off.Visible = False
Form1.Reley01_on.Enabled = False
Form1.Reley01_on.Visible = False
Form1.Reley02_on.Enabled = False
Form1.Reley02_on.Visible = False
Form1.Reley03_on.Enabled = False
Form1.Reley03_on.Visible = False
Form1.Photo01_on.Enabled = False
Form1.Photo01_on.Visible = False
Form1.Photo01_off.Enabled = False
Form1.Photo01_off.Visible = False
Form1.IRemitter01_off.Visible = False
Form1.IRemitter01_off.Enabled = False
Form1.IRemitter01_on.Visible = False
Form1.IRemitter01_on.Enabled = False
Form1.work
End Sub

Private Sub Photo01_off_Click()
Form1.IRcmnd_on.Visible = False
Form1.IRcmnd_off.Visible = True
End Sub

Private Sub Photo01_on_Click()
Form1.wrem.Caption = "            "
Form1.MSComm1.Output = "C14$0S"
For i = 0 To 10000000
Next i
strAnsw = Form1.MSComm1.Input
If strAnsw = "C14$0SC14001" Then
Form1.IRcmnd_off.Visible = False
Form1.IRcmnd_on.Visible = True
Form1.wrem.Caption = strAnsw
Else
Form1.IRcmnd_on.Visible = False
Form1.IRcmnd_off.Visible = True
Form1.wrem.Caption = strAnsw
End If
If strAnsw = "C14$0SC14129" Then
Form1.Lamp_on.Visible = True
Form1.Lamp_off.Visible = False
```

```
Form1.wrem.Caption = strAnsw
End If
End Sub

Private Sub Reley01_off_Click()
Form1.MSComm1.Output = "R14$0F"
Form1.MSComm1.Output = "R14$0S"
For i = 0 To 10000000
Next i
strAnsw = Form1.MSComm1.Input
If strAnsw = "R14$0FR14$0SR14#0F" Then
Form1.Lamp_on.Visible = False
Form1.Lamp_off.Visible = True
End If
End Sub

Private Sub Reley01_on_Click()
Form1.MSComm1.Output = "R14$0N"
Form1.MSComm1.Output = "R14$0S"
For i = 0 To 10000000
Next i
strAnsw = Form1.MSComm1.Input
If strAnsw = "R14$0NR14$0SR14#0N" Then
Form1.Lamp_off.Visible = False
Form1.Lamp_on.Visible = True
End If
End Sub

Private Sub Reley02_off_Click()
Form1.MSComm1.Output = "R14$1F"
Form1.MSComm1.Output = "R14$1S"
For i = 0 To 10000000
Next i
strAnsw = Form1.MSComm1.Input
Form1.wrem.Caption = strAnsw
If strAnsw = "R14$1FR14$1SR14#1F" Then
Form1.Torsher_on.Visible = False
Form1.Torsher_off.Visible = True
End If
End Sub

Private Sub Reley02_on_Click()
Form1.MSComm1.Output = "R14$1N"
```

```
Form1.MSComm1.Output = "R14$1S"
For i = 0 To 10000000
Next i
strAnsw = Form1.MSComm1.Input
If strAnsw = "R14$1NR14$1SR14#1N" Then
Form1.Torsher_off.Visible = False
Form1.Torsher_on.Visible = True
End If
End Sub

Private Sub Reley03_off_Click()
Form1.MSComm1.Output = "R14$2F"
Form1.MSComm1.Output = "R14$2S"
For i = 0 To 10000000
Next i
strAnsw = Form1.MSComm1.Input
If strAnsw = "R14$2FR14$2SR14#2F" Then
Form1.Bra_on.Visible = False
Form1.Bra_off.Visible = True
End If
End Sub

Private Sub Reley03_on_Click()
Form1.MSComm1.Output = "R14$2N"
Form1.MSComm1.Output = "R14$2S"
For i = 0 To 10000000
Next i
strAnsw = Form1.MSComm1.Input
If strAnsw = "R14$2NR14$2SR14#2N" Then
Form1.Bra_off.Visible = False
Form1.Bra_on.Visible = True
End If
End Sub
```

Подведем итоги

Первая версия системы завершена.

Конечно, даже после отладки модулей в MPLAB пришлось обратиться к работе с макетом. Думаю, это в первую очередь, связано с тем, что несколько разработок – это еще не опыт работы. Но можно надеяться на изменение ситуации со временем. Полезный вывод из сделанных ошибок – при

программировании на языке С труднее учитывать реальное время. Например, устанавливая задержку с помощью оператора цикла `for`, который на С выглядит, как один оператор, не следует забывать, что на ассемблере операторов будет значительно больше. Наконец, еще один вывод – работать с демоверсиями – это создавать себе дополнительные проблемы.

Тем не менее, план, который мы наметили для первой части работы, реализован: есть модули и среда программирования системы, пусть не столь красивая и удобная, как в коммерческих реализациях, но работающая.

В целях удешевления первых экспериментов (я уже говорил об упрощении программатора) настоятельно рекомендую использовать одну макетную плату. Моя макетная плата кроме интерфейсной микросхемы, стабилизатора напряжения и панельки под контроллер имеет три красных светодиода АЛ307 и фотоприемник TSOP с его «атрибутами». Макетная плата позволила отладить все три модуля. При этом светодиод АЛ307 я использовал в качестве излучателя ИК-команд.

Что касается внешнего вида готовых образцов, если вы решите их сделать: значительная часть оборудования может располагаться в шкафах, силовых или слаботочных, то есть быть скрытой от глаз. Кроме того, сегодня можно купить подходящие коробки и коробочки вполне приятного «товарного» вида. В частности, есть коробки с макетными платами нескольких размеров, предназначенные для установки на DIN-рейку (то есть в шкаф). Они имеют весьма респектабельный вид и удобны для сборки модулей. А этикетки можно очень удачно печатать на принтере. Так что получить вполне приемлемый внешний вид для своих разработок по силам любому. Тем более что и промышленные образцы зачастую имеют очень «прозаический», хотя и хорошо исполненный, конструктив.

Возвращаясь к предисловию, добавлю небольшое послесловие к первой части. Его можно назвать «Ода ошибкам».

Ода ошибкам

Любого из нас ошибки сердят, вызывают раздражение, могут подорвать веру в собственные силы. Не принято в книге,

источнике знаний, делать множество ошибок. Но, если вдуматься, не наши ли ошибки, как наши глаза и уши, ведут нас к знаниям? Как палка слепого, они позволяют определять препятствия, заставляют задуматься, правильным ли путем двигаешься? Собственные ошибки запоминаются лучше и служат дольше. Когда я попадаю в безвыходную ситуацию, когда отсутствие решения подобно беспросветному мраку, мне помогает простой прием – я не пытаюсь найти правильное решение, я принимаю заведомо ошибочное. С ним я работаю до того момента, когда оно либо трансформируется в правильное решение, либо подсказывает путь к нему. Самое плохое, что поджидает меня в этом случае, – напрасно потраченное время. Но напрасно ли оно потрачено? Двигаясь неверным путем, я все-таки делаю многое, что позже применяю в других случаях, но в схожих ситуациях. Со временем, с накопленным опытом приходит интуитивное видение решений. Но это относится, в первую очередь, к узкой области деятельности. Стоит выйти за пределы этой области – и вашей интуиции понадобится тросточка, чтобы пройти весь путь без болезненных падений. Я не люблю ошибаться, не люблю признаваться в ошибках. Я вообще не люблю ошибки! Но понимаю их пользу, как необходимость и пользу горьких лекарств.

Во второй части книги мы разработаем еще несколько полезных модулей. Возможно, «причешем» уже готовые. Возможно, создадим среду программирования больше похожую на профессиональную версию. Посмотрим...



Глава 2

Как расширить систему

В этой части книги я предлагаю расширить набор модулей. В первую очередь, за счет разработки модуля цифровых вводов.

Почему я не включил модуль цифровых вводов в первую часть?

Модуль, в основном, предназначен для подключения датчиков с «сухими контактами» или аналогичным выходом, имеющих два состояния – включено и выключено. Это противопожарные датчики, датчики охраны. К модулю цифровых вводов можно подключить пороговые датчики освещенности: освещенность достигла некоторой величины – датчик замыкает контакты; освещенность упала – датчик выключает контакты. Такими могут быть датчики температуры, влажности, давления, положения, термостаты и т.д. Большой интерес в этом ряду представляют датчики движения. С их помощью легко организовать автоматическое управление светом в коридоре, прихожей или на лестнице.

Часть датчиков, например датчики освещенности и температуры, достаточно просто изготовить самостоятельно. Но пирометрические датчики движения даже в собственном исполнении могут оказаться слишком дорогими. Использовать противопожарные или охранные датчики в самостоятельном исполнении я не вижу смысла – они должны быть очень надежны, так как требования к устройствам, обрабатывающим их сигналы, очень жесткие. Кроме того, если вам захочется

разработать устройства подобного назначения, то не составит труда сделать это самостоятельно. Однако я не рекомендовал бы возлагать большие надежды на использование самодельных противопожарных и охранных устройств.

Тем не менее, мы разработаем модуль цифровых вводов, а в приложении я постараюсь привести хотя бы одну схему датчика движения.

Расширение состава модулей, кроме модуля цифровых вводов, осуществим за счет замены реле в релейном модуле на аналог твердотельного реле, состоящий из оптопары и триака. Подобный модуль, рассчитанный на подключение активной нагрузки ламп накаливания, хорошо подойдет для включения и выключения света. Мощность нагрузки может составлять 500–600 Вт, а габариты позволяют встроить модуль в обычный выключатель света. Попутно мы рассмотрим возможность плавной регулировки яркости света.

Использование релейного модуля для подключения музыкального центра к громкоговорителям, установленным в других помещениях, решает вопрос о распределении звука. Но в ситуации, когда вы намерены в другом помещении не только слушать музыку, но и смотреть фильмы, полезным может оказаться аудиокоммутатор. О модуле подобного назначения мы тоже поговорим. Посмотрим, нельзя ли его применить и для коммутации видеосигнала.

Конечно, все описанные устройства имеют основное назначение – исследование возможностей микроконтроллеров. Предполагается, что они будут собраны на макетной плате, разложены на столе, а по окончании экспериментов отправятся в ящик с деталями для разборки «по случаю». Вместе с тем, кому-то, возможно, приглянется идея применить разработки в своей комнате или квартире. Если в квартире не заложено большого количества проводов для организации системной сети, прокладывание проводов может представлять некоторые сложности. Есть несколько решений, позволяющих избежать этого при организации системной сети. Я уже упоминал протокол X10 – использование силовых проводов в качестве «среды обитания» системных сигналов. При выборе этого решения настоятельно рекомендую воспользоваться

готовыми изделиями, например, обратившись в фирму «Умный дом». Другое решение – радиоканал или использование инфракрасного излучения для передачи сетевых сигналов. Эти возможности мы обсудим.

И, конечно, постараемся расширить средства управления. Добавить к компьютеру и старому пульту от телевизора что-то, не менее успешно решающее эти задачи.

Модуль цифровых вводов

Зачем нужен модуль цифровых вводов, я уже говорил. Что же он собой представляет в плане постановки задачи? Модуль должен иметь некоторое количество входов, каждый из которых может быть замкнут на общий провод или разомкнут. В ответ на запрос центрального управляющего устройства модуль передает состояние своих входов. К входам присоединяются датчики. Кроме уже упомянутых это могут быть датчики, регистрирующие состояние бытовой аппаратуры. Например, датчик состояния телевизора, регистрирующий, включен он или выключен. Применение подобного датчика особенно актуально, если телевизор, как это зачастую получается, управляется с помощью ИК-команд. Поясню.

Стандартная ситуация: систему можно построить таким образом, что управление будет происходить по времени суток. В час ночи, если вы ложитесь спать раньше, система может выключить все бытовые приборы, весь свет в доме (или квартире), всю аудио- и видеоаппаратуру, которые могли остаться включенными. Однако есть одно «но». ИК-сигнал выключения некоторых телевизоров, многих музыкальных центров и видеомагнитофонов полностью совпадает с сигналом включения. Если одна из команд прошла мимо системы, вместо выключения устройства вы включите его. Обычно в программе управления можно устанавливать флаги состояния. Каждое включение телевизора устанавливает флаг **TV_ON**, а выключение сбрасывает его. А если телевизор оказался выключен из сети? Флаг будет успешно устанавливаться и сбрасываться, не решая проблемы.

Можно применить релейный модуль, подключая телевизор, музыкальный центр и остальное оборудование к сети

через контакты реле. Но хотелось бы иметь запасной вариант. Таким вариантом станет датчик, который фиксирует ток в проводах подключения бытовой техники. Лучше использовать, например, датчик Холла для определения тока в цепи, но можно применить и что-нибудь попроще. Об этом мы поговорим ниже, но подобный датчик подключить к системе разумнее всего с помощью модуля цифровых вводов или сам модуль сделать частью датчика.

Есть очень удобные и дешевые датчики, которые называют «герконовыми». Они удобны для установки на двери или окна. На основе такого датчика можно организовать автоматическое включение и выключение света в ванной или туалете. При первом открывании двери свет включается, при втором выключается. «Герконовый» датчик, конечно, к системе подключается через модуль цифровых вводов. Если установить датчик на входную дверь, а в основной программе поставить счетчик, можно организовать подпрограмму определения момента, когда все покидают квартиру или дом. Этот момент может быть отправной точкой для принудительного отключения от сети всех электроприборов и перекрывания вводов воды во избежание протечек. Или включения системы имитации присутствия людей в доме, что используется, как часть подсистемы охраны. Позже, когда кто-то приходит домой, такой датчик можно использовать для организации сцены обслуживания, которую я описал в предисловии как «Возвращаюсь я с работы...». Таким образом, модуль цифровых вводов вполне можно отнести к базовым и обязательным модулям системы «Умный дом».

Я хочу обсудить еще одно применение модуля цифровых вводов – в качестве интерфейса к клавишным устройствам управления. Имея 8 входов, подобный модуль может работать с устройством управления, снабженным 8 клавишами. Этого достаточно для многих задач управления. Если же использовать матрицу 4×4 для построения клавиатуры, количество подаваемых команд увеличится до 16, а количество используемых выводов порта останется равным восьми.

Модуль цифровых вводов легко сделать из программы для релейного модуля. Перенаправим все выходы, предназначенные

для подключения реле, на вход. Будем отслеживать состояние всех входов и записывать их. Остается только передавать их состояние в ответ на запросы главной программы, как это было с системными ИК-командами в модуле приема ИК-кодов.

Присвоим командам модуля цифровых входов префикс, например D, и получим формат запроса:

```
Dxx$0S
```

где xx – два символа номера модуля от 0 до 15, а 0 после $ – «заставка» для поддержки формата.

Формат ответа модуля тоже, очевидно, будет:

```
Dxxyyy
```

где yyy – символьное представление десятичного числа, отображающего состояние входов (как это было сделано для приемника ИК-команд).

Программа модуля цифровых вводов на языке С

Входы RA0–RA2 я оставил для выхода, к ним подключены индикаторы.

Файл заголовка

```
void putch(unsigned char);
unsigned char getch(void);
int init_comms();
int sim_num_adr();
int cmd();
int din_stat();
```

Основной файл

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "digin.h"

unsigned char input;          // Для считывания приемного
регистра.
unsigned char MOD_SIM1;       // Первый символ адреса модуля.
unsigned char MOD_SIM2;       // Второй символ адреса модуля.
```

```
unsigned char command_reciev [6]; // Массив для полученной
команды.
int MOD_ADDR;            // Заданный адрес модуля, как число.
int sim_end_num = 0;          // Полный символьный номер
модуля.
int MOD_NUM;          // Полученный адрес модуля, как число.
unsigned char DINSTAT = 0xF8; // Статус позиционно: 1 - вкл,
0 - выкл.
unsigned char DINSIM1;      // Символы состояния для передачи.
unsigned char DINSIM2;
unsigned char DINSIM3;
int i;
                // Получение байта.
unsigned char getch()
{
   while(!RCIF)       // Когда регистр не пуст.
      continue;
   return RCREG;
}
                // Вывод одного байта.
void putch(unsigned char byte)
{
   while(!TXIF)       // Когда регистр пуст.
      continue;
   TXREG = byte;
}

// Преобразуем символьный адрес в число.
int sim_num_adr()
{
      sim_end_num = 0;
      MOD_SIM1 = command_reciev [1]; // Первый символ номера.
      MOD_SIM2 = command_reciev [2]; // Второй символ номера.
      MOD_SIM1 = MOD_SIM1 - 0x30;
      MOD_SIM2 = MOD_SIM2 - 0x30;
      sim_end_num = MOD_SIM1*0x0A + MOD_SIM2;
      return sim_end_num;
}

// Получение и выполнение команды.
int cmd()
{
   switch (command_reciev [5]) {
```

```
        case "S": din_stat();
        break;
        }
}

// Выполнение команды передачи состояния.
int din_stat()
{
int din;
int din1;
int din2;
int din3;

    din = DINSTAT;          // Преобразуем состояние в символы.
    din1 = din/0x64;
    DINSIM1 = din1 + 0x30;
    din2 = (din - din1*0x64)/0xA;
    DINSIM2 = din2 + 0x30;
    din3 = (din - din1*0x64 - din2*0xA);
    DINSIM3 = din3 + 0x30;

    command_reciev[0] = "D";
    command_reciev[1] = MOD_SIM1+0x30;
    command_reciev[2] = MOD_SIM2+0x30;
    command_reciev[3] = DINSIM1;
    command_reciev[4] = DINSIM2;
    command_reciev[5] = DINSIM3;

        CREN =0;     // Запрещаем прием.
        RB0 = 1;     //Переключим драйвер RS485 на передачу.
        TXEN = 1;       // Разрешаем передачу.
        for (i=0; i<6; ++i)   putch(command_reciev[i]);
        for (i=0;i<1000;i++);     // Задержка для вывода.
        for (i=0; i<6; ++i) command_reciev [i] = " ";
        RB0 = 0;     // Выключаем драйвер RS485 на передачу.
        TXEN = 0;    // Запрещаем передачу.
        CREN =1;     // Разрешаем прием.
        RA1 = 0;
        DINSTAT = 0xF8; // Сбросим состояние.
}

int init_comms()          // Инициализация модуля.
{
```

```
PORTA = 0xF8;                // Настройка портов A и B.
CMCON = 0x7;
TRISA = 0xF8;
TRISB = 0xFE;
RCSTA = 0b10010000;            // Настройка приемника.
TXSTA = 0b00000110;            // Настройка передатчика.
SPBRG = 0x68;                // Настройка режима приема-передачи.
RB0 = 0;            // Выключаем драйвер RS485 на передачу.
RA0 = 1;
}

void main(void)
{
// Инициализация модуля.
   init_comms();
   for (i=0; i<6; ++i) command_reciev [i] = " ";
   command_reciev [0] = "D";
//Прочитаем и преобразуем номер модуля.
   MOD_ADDR = PORTB;      // Номер модуля в старших битах.
   MOD_ADDR = MOD_ADDR>>4;  // Сдвинем на четыре бита.

// Начинаем работать.
start: CREN =1;
   if ((PORTA&0xF8) != 0xF8)
   {
      DINSTAT = DINSTAT&PORTA;
      RA1 = 1;
   }

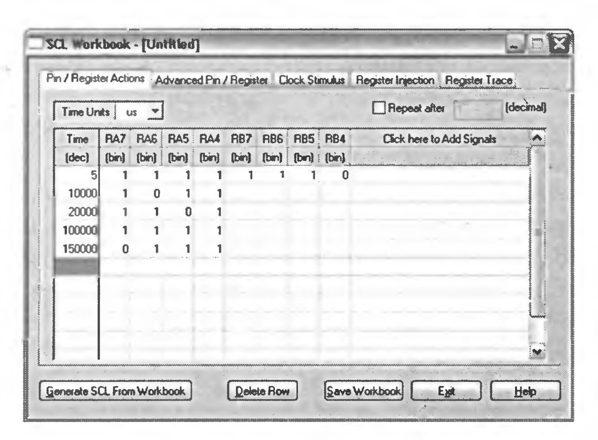
   if (RCIF)       // Если пришел запрос по сети.
      input = getch();
      switch (input)
   {
      case "D":    // Если обращение к модулю.
      for (i=1; i<6; ++i)   // Запишем команду в массив.
         {
         input = getch();
         command_reciev [i] = input;
         }
      MOD_NUM = sim_num_adr();     // Чтение из сети.

      if (MOD_NUM != MOD_ADDR) break;    // Если не наш адрес.
      else
```

```
        if (command_reciev [3] = "$") cmd();   //Если команда.
        default: goto start;
    }
    goto start;
}
```

Выводы порта A RA0–RA2 на время работы с макетом я программирую так, чтобы индикатор на выводе RA0 включался при включении модуля, а индикатор на выводе RA1 отображал наличие изменений состояния входов модуля.

При настройке программы в MPLAB я использую данные, показанные на рис. 2.1.



Time (dec)	RA7 (bin)	RA6 (bin)	RA5 (bin)	RA4 (bin)	RB7 (bin)	RB6 (bin)	RB5 (bin)	RB4 (bin)
5	1	1	1	1	1	1	1	0
10000	1	0	1	1				
20000	1	1	0	1				
100000	1	1	1	1				
150000	0	1	1	1				

Рис. 2.1. Отладочные данные

В этом файле многие переменные и функции я оставил в обозначениях, близких к исходным. Я уже говорил, что собираюсь переделать программу из программы релейного модуля. Часть текста, подправив, я взял из текста программы модуля приемника системных ИК-команд. Правка была минимальной.

Здесь мне хотелось бы еще раз вернуться к вопросу о написании текста программы. Многие переменные в представленном выше тексте обозначены так, что трудно понять, для каких целей они служат. Функции не облегчают понимание их назначения. Вернувшись к тексту после нескольких недель

перерыва, начинаешь понимать, что лучше было бы все сделать иначе. Удобно обозначать переменные, чтобы легко было понять их назначение. То же касается и функций. Для этого есть много правил, которые лучше поискать в руководствах по программированию. Но есть один момент в работе над текстом. Переменную удобно обозначить, например, `int digital_input_close`. Однако, используя эту переменную десятки раз, переделывая программу столько же раз, ты понимаешь, что каждый раз вписывать это многословное название переменной – утомительное занятие. Можно, конечно, копировать название из текста программы. Но для этого приходится возвращаться к началу, затем искать место, где ты работал. Можно поступить иначе, и это будет полезно во всех отношениях – использовать файл todo.txt. В нем перечислить все переменные и функции, попутно описав их. Переключение между файлами быстрое, а копирование не сложное, поскольку перечень переменных и функций можно сделать коротким.

При проверке макета (а для его проверки я использую тестовую часть основной программы, исправив обращение к одному из модулей на обращение к модулю цифровых входов – `D14$0S`), первое, что мне не нравится, но не является неожиданностью, – это поведение входов. Думаю, причина в отсутствии «подтягивающих» резисторов.

Я впаиваю на макет подтягивающие резисторы (резисторы между плюсом питающего напряжения и выводом) на входы, и картина действительно меняется – макет полностью перестает реагировать на тестовые команды!

Создается впечатление, что или конвертер RS232–RS485, или интерфейс RS485 макета вышли из строя. В моем распоряжении только мультиметр. Не самый удачный вариант. Первое, что приходит в голову после нескольких неудачных попыток оживить модуль, – внести в программу изменение: при инициализации переключить драйвер RS485 на передачу (бит RB0).

Бит переключается, что уже хорошо. Затем, манипулируя индикаторами на выводах RA0–RA2, я начинаю проверять, в чем дело? И проблема оказывается в простой халатности – при

переделке программы релейного модуля я, хотя и намеревался, не изменил основную часть программы, которую собирался позаимствовать от программы фотоприемника. После изменения программы модуль цифровых входов стал работать вполне убедительно.

В тексте выше приведена уже исправленная версия программы.

Теперь осталось проверить работу модуля в основной программе в режиме **Работа**. В этом режиме программа не реагирует ни на что, кроме сетевых событий. Ранее для выхода из режима **Работа** я использовал ИК-команду. Сейчас я хочу использовать изменение входа RA3, внеся для этой цели изменения в основную программу:

```
Sub work()
Do

Form1.KeyPreview = True
Form1.MSComm1.Output = "D14$0S"
For i = 0 To 10000000
Next i
strAnsw = Form1.MSComm1.Input  и т.д. до строки

Case "D14$0SD14240"
ext = True
End Select
```

Таким образом, я поменял обращение к модулю (было обращение к модулю фотоприемника) и условие выхода.

Программа работает.

HEX-файл модуля цифровых вводов

Выводы RA0–RA2 используются в «макетном варианте» для индикации!

```
:1000000830100308A000428203084004030162000C
:10001000830103300C200FF30C10040308400413012
:100020001A2083019E2A04068001840A0406031D07
:100030001328003 4F00026208000840A040870068B
:1000400003190034 1B2883120313C100C21B31287B
```

:10005000421B392842088A004108C10A0319C20A12
:100060008200831342188317410 8C10A84000008E4
:0200700080086
:1004860083018C1E432A1A080800830134 08533A54
:10049600031D0800FC2A8301BF00831203130C1EF0
:1004A600502A3F0899000800F401F5010310F30CE7
:1004B600F20C031C652A7008F40771080318710A08
:1004C600F5070310F00DF10D720873040319 0034DB
:1004D600592AF83083018500073 09F00F8308316CB
:1004E6008500FE308600903083129800063083 1611
:1004F6009800683099008312061005140800 8301DD
:10050600AD01AE013008A4003108A500D030A40723
:10051600A5070A30F200F3012408F000F101572282
:10052600250874 07AD0075080318750AAE00F100BA
:100536002D08F00008006C22AB01AC01AB2A2B0899
:100546002F3E8400831320308000AB0A0319AC0AC7
:100556002C08803AF0008030700206300319 2B0216
:10056600031CA22A4430AF000608A700A8010430E5
:10057600F000280DA80CA70CF00BBC2AEA2AAB0148
:10058600AB0AAC012C08803AF000803070020630CD
:1005960003192B020318DB2A4322A6002B082F3E41
:1005A6008400831326088000AB0A0319AC0AC52A07
:1005B60082227008A9007108AA002806031DE52AF0
:1005C60027082906031DEA2A2430B20048221816F5
:1005D6000508F839F83A0319F32A0508C005851401
:1005E6008C1EF72A4322A600260844 3A0319C22A7B
:1005F600EA2A83014008B900BA016430F200F30127
:100606003A08F1003908F000B0237408BB007508F9
:10061600BC003B08303EA0006430F200F3013C0809
:10062600F1003B08F00057223A08F1003908F000C3
:100636007408F002031CF1037508F1020A30F20097
:10064600030F301B0237408B7007508B800370806
:100656 00303EA1000A30F200F3013808F1003708F5
:10066600F00057227408BD007508BE006430F20021
:10067600F3013C08F1003B08F00057223A08F1006C
:100686003908F0007408F002031CF1037508F10242
:100696003D08F002031CF1033E08F1027008B500A4
:1006A6007108B6003508303EA2004430AF00240879
:1006B600303EB0002508303EB1002008B2002108C7
:1006C600B3002208B40018120614831698168 31273
:1006D600AB01AC01772B2B082F3E8400831300085 7
:1006E6004E22AB0A0319AC0A2C08803AF00080307F
:1006F600700206300319 2B02031C6E2BAB01AC01F2

```
:100706002C08803AF00083307002E83003192B027F
:10071600031891 2BAB0A0319AC0A832BAB01AC016E
:100726002C08803AF0008030700206300319 2B0244
:10073600031 8A72B2B082F3E84008313203080003C
:10074600AB0A0319AC0A932B06108316981283127 0
:1007560018168510F830C0000800F601F11FBA2BF4
:10076600F009F00A0319F103F1097617F617730871
:10077600803 9F606F31FC62BF209F20A0319F303B2
:10078600F309C62BF601F401F5017208730 4031D83
:10079600CF2BF001F10100341F30F6040310F60AE6
:1007A600F20DF30D031CD22BF30CF20C730871023D
:1007B600031DDF2B72087002031CE72B7208F00280
:1007C6007308031C730AF102F40DF50DF60BF61A05
:1007D600D72BF61FF32BF409F40A0319F503F509D1
:1007E6007408F2007508F300761F0034F009F00A69
:0A07F6000319F103F1090034F8348F
:00000001FF
```

Пришло время поговорить о датчиках, с которыми можно провести эксперименты, используя модуль цифровых вводов.

Я уже говорил, что наиболее интересными датчиками будут датчики движения, противопожарные датчики, датчики температуры (пороговые измерители температуры), датчики положения (в частности, герконовые).

Рассмотрим их по порядку, начиная с простейшего – герконового датчика.

Герконовый датчик (название происходит от сокращения «герметизированные контакты») представляет собой пару: магниточувствительные контакты в стеклянной колбе и постоянный магнит. Реальный датчик, конечно, имеет пластмассовые корпуса и для контактов, и для магнита. Когда контакты попадают в поле постоянного магнита, они замыкаются (или переключаются для варианта с контактами на переключение). Естественно, соединив один вывод контактов с выводом модуля цифровых вводов, а другой – с общим проводом модуля, мы получаем сигнал о взаимном положении контактов и магнита. Если контакты укрепить на дверной короб, а магнит – на дверь, при закрытой двери контакты замкнутся, а при открытой разомкнутся. Этот сигнал можно использовать в основной программе, чтобы инициировать любые события.

Положим, мы установили герконовый датчик на входную дверь. Теперь вполне можно реализовать программу «Возвращаюсь я с работы...», о которой я говорил в предисловии. Напомню: «Возвращаюсь я с работы. Система встречает меня – зажигает свет в прихожей, кипятит воду для кофе (конечно, чайник я наполняю утром). А когда я перехожу в гостиную с чашкой кофе, она включает телевизор на моей любимой программе новостей, чтобы я, усевшись в любимое кресло, посмотрел, что произошло в мире за день. Свет на кухне и в прихожей, который я, конечно, забыл выключить, она выключит сама».

Для реализации подобной программы все, собственно, есть.

Можно установить герконовые датчики на дверь туалета, а основную программу построить так, чтобы при первом открывании двери свет в туалете включался, а при втором открывании выключался. Естественно, в систему необходимо добавить релейный модуль для управления светом или модуль с триаком, о котором речь пойдет ниже.

Герконовые датчики, несмотря на всю свою простоту, могут найти множество применений в экспериментах с системой «Умный дом».

Рассмотрим, как можно сделать датчик температуры. В качестве датчика температуры можно использовать терморезистор. Сделав делитель из терморезистора и обычного резистора, подключив напряжение к делителю, можно снимать сигнал напряжения на терморезисторе при изменении температуры. Теперь осталось подключить компаратор и реле, чтобы при достижении некоторого значения напряжения реле включалось (с модулем цифровых вводов можно использовать вместо реле транзистор с открытым коллектором). Используя в качестве обычного резистора пару из переменного и постоянного резистора, можно изменять заданное напряжение срабатывания. Подобных схем великое множество, при необходимости вы можете использовать любую. Можно построить и свою схему, в основе которой может лежать чувствительность к температуре не только терморезистора, но транзистора и т.д.

Как в системе можно использовать подобный датчик? Например, можно настроить датчик на срабатывание при температуре ниже –15° и поместить термочувствительный элемент за окном. В качестве индикатора «похолодания» можно использовать свет в прихожей. Действия программы будут следующими: температура за окном упала ниже –15°. Когда открывается входная дверь, снабженная герконовым датчиком, свет в прихожей выключается и включается вновь – не забудьте теплое пальто!

Я уже говорил, что, управляя бытовой аппаратурой с помощью ИК-кодов, сталкиваешься с проблемой определения ее текущего состояния – включен телевизор или выключен? Здесь тоже можно использовать датчик температуры для безусловного выключения телевизора в заданное время, поскольку включенный телевизор греется, а выключенный нет.

Датчики движения. Насколько я понимаю, они появились как датчики в составе охранных систем. Есть несколько разновидностей подобных датчиков. С моей точки зрения, наиболее интересны пирометрические датчики движения. Но они достаточно дороги. Более дешевыми могут оказаться микроволновые датчики.

Есть еще один вид датчиков, которые могут найти практическое применение даже при самостоятельном изготовлении. Это датчики протечек. Датчики, реагирующие на появление воды там, где ее не должно быть. Установив подобные датчики возле батарей отопления, в местах ввода горячей и холодной воды, возле стоков раковин, можно избежать серьезных неприятностей при появлении протечек воды. Сигнализируя об аварии, система выручит вас на этапе, когда еще не поздно перекрыть вентили или перестать пользоваться раковиной и вызвать специалиста для устранения неисправности. Хотя надежнее, я думаю, будут все-таки готовые датчики.

Есть еще одно применение модуля цифровых вводов, о котором я упоминал раньше, – системное устройство управления.

Само системное устройство управления – это клавишный пульт. Контакты при нажатии клавиши замыкают один из входов модуля на общий провод. Восемь входов модуля позволяют

сделать простой пульт с восьмью клавишами, что позволяет подать восемь команд. Достаточно ли этого? Достаточно для управления всеми светильниками в комнате. Или всеми основными светильниками в квартире. Этого хватит для управления одним аудио- или видеоустройством, например видеомагнитофоном. Но управлять телевизором будет неудобно, поскольку трудно будет ввести номер канала. Для ввода номера канала обычные пульты имеют цифровую клавиатуру с цифрами от 0 до 9. Чтобы увеличить количество клавиш управления, достаточно добавить кнопки не с одним контактом на замыкание, а с двумя. Если первые восемь кнопок будут менять состояние одного бита, то остальные – состояние двух бит одновременно. Для случая использования модуля с клавиатурой в программу контроллера следует добавить задержку на 10–50 мс перед определением состояния входов:

```
if ((PORTA&0xFF) != 0xFF)
   {
   for (i=0; i<1000; i++);
   DINSTAT = DINSTAT&PORTA;
   }
```

Необходимость в задержке связана с «дребезгом» контактов и неодновременным замыканием контактов у двухконтактных кнопок.

Применение кнопок с двумя замыкающими контактами, то есть замыкающими первый вход и второй, первый и третий и т.д. одновременно, позволит добавить на пульт еще семь клавиш управления. Одновременное замыкание второго и третьего, второго и четвертого и т.д. – добавит еще шесть клавиш. Даже 21 клавиши управления для пульта, мне кажется, более чем достаточно для любых применений этого решения к построению клавишного пульта управления.

Позже мы обсудим вопрос о том, что можно сделать в экспериментах, если в квартире не заложено достаточно большого количества проводов, а менять проводку нет возможности. Я имею в виду использование радиоканала. Это же решение можно применить и к пульту управления, построенному на основе модуля цифровых вводов, для превращения его в переносной пульт управления.

Резюме. Мы расширили возможности системы за счет нового устройства – модуля цифровых вводов. Рассмотрели и несколько применений этого модуля. И, как мне кажется, с появлением нового модуля система приобрела новое качество.

Модуль с триаком

Релейный модуль, описанный в самом начале, универсален во многих отношениях. Он позволяет, выбрав соответствующее реле, коммутировать настольные лампы и электрический чайник, переключать громкоговорители и коммутировать входы усилителя (с не очень высоким качеством).

На базе релейного модуля можно изготовить выключатель света, который в состоянии заменить обычный. Конечно, в этом случае потребуется подвести провода системной сети. Но, если говорить о замене обычного выключателя света на системный, лучше изменить способ управления светильником. Конечно, если это не лампа дневного света, которую рекомендуется включать с помощью реле.

Есть хороший коммутатор для ламп накаливания – триак. Триак, или семистор – управляемый полупроводниковый прибор, который в отличие от тиристора может использоваться в цепях переменного тока. При мощности нагрузки до 500–600 Вт ему достаточно небольшого радиатора в виде металлической пластины по размерам выключателя света. Задача управления этим коммутатором тока, при всей ее видимой простоте, может оказаться весьма интересной. Очень хорошее решение эта задача получила с появлением оптопары – светодиод и фототриак. Подобные микросхемы, насколько я знаю, выпускает фирма Toshiba в серии TPL (TPL3061, TPL3041) и несколько производителей – серия MOC (рис. 2.2.и 2.3).

Светодиод оптрона включается в модуль управления светом, как включаются индикаторы, при этом полностью изолируя модуль от силовой сети. Триак оптрона подключается к управляющему электроду мощного триака. Стоимость же микросхемы и триака может оказаться меньше, чем стоимость реле.

Схема модуля с триаком может выглядеть, как показано на рис. 2.4.

Тиристорные оптопары						
Наименование	Схема №	Uкомм., пик., В	Iсраб., вх., мА	Zero-Cross*	Uиз., кВ	Тип корпуса
MOC3020	10					PDIP6
MOC3021	10	400	15		7,5	PDIP6
MOC3023	11	400	5		7,5	PDIP6
MOC3041	11	400	15	*	7,5	PDIP6
MOC3042	11	400	10	*	7,5	PDIP6
MOC3061	11	600	15	*	7,5	PDIP6
MOC3062	11	600	10	*	7,5	PDIP6
MOC3063	11	600	5	*	7,5	PDIP6

Рис. 2.2. Тиристорные оптопары

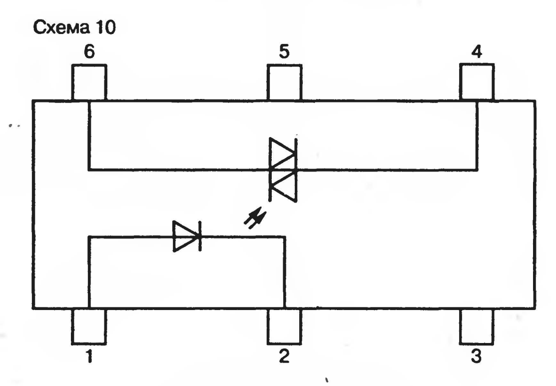


Рис. 2.3. Оптопара для модуля

С моей точки зрения, подобного выключателя достаточно для системы. Но применение триака в качестве коммутатора для управления светом открывает возможность плавной регулировки яркости света. Для тех, кого заинтересуют эксперименты в этой области, можно предложить следующее решение.

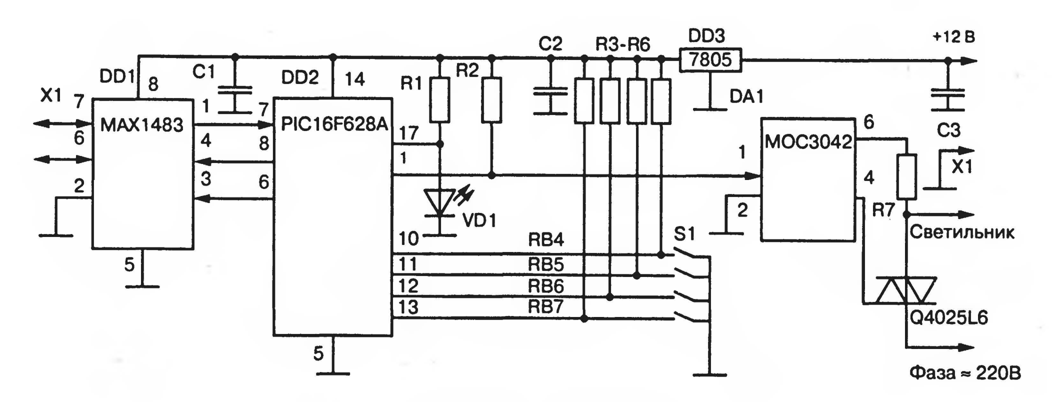


Рис. 2.4. Схема модуля с триаком

Модуль с плавной регулировкой яркости

Плавная регулировка яркости света триаком связана с тем, что яркость ламп накаливания зависит от эффективного напряжения. Чем оно ниже, тем меньше яркость. Устройства подобного типа имеют название «диммер».

Тиристор или триак открывается сигналом, подаваемым на управляющий электрод. Осциллограмма напряжения на нагрузке, лампе накаливания представлена на рис. 2.5.

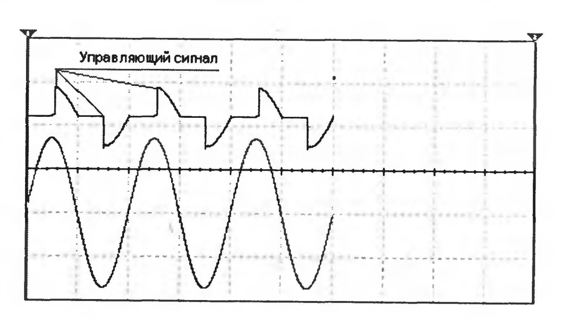


Рис. 2.5. Осциллограмма напряжения при тиристорном управлении

В верхней части отображено напряжение, интересующее нас, а внизу – переменное силовое напряжение 220 В.

При подаче управляющего сигнала триак открывается, и напряжение полностью падает на нагрузке. При переходе напряжения через ноль триак закрывается. Напряжение на лампе накаливания появится только после прихода следующего управляющего сигнала. В данном случае частота управляющих сигналов 100 Гц. Усеченное синусоидальное напряжение на лампе накаливания в данном случае равнозначно подключению лампы к напряжению ~110 В. Яркость лампы уменьшается.

Сдвигая момент подачи управляющего сигнала ближе к моменту перехода напряжения через ноль (рис. 2.6 – маркерами помечены моменты подачи управляющего сигнала), мы получим большую яркость. Удлиняя время подачи управляющего сигнала после перехода напряжения через ноль (рис. 2.7), мы получим меньшую яркость свечения лампы.

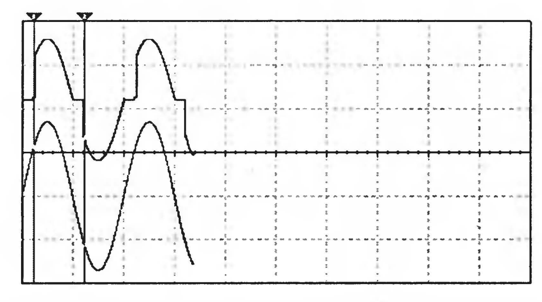


Рис. 2.6. Осциллограмма напряжения на светильнике при увеличении яркости

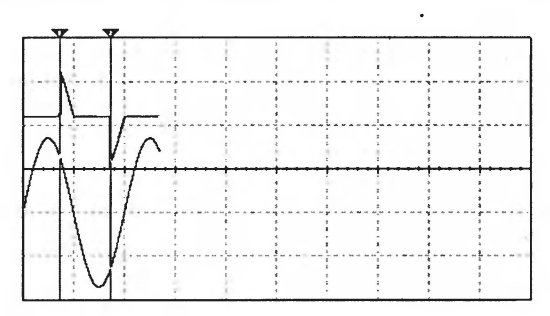


Рис. 2.7. Осциллограмма напряжения на светильнике при уменьшении яркости

При построении модуля следует включать светодиод оптрона в нужные моменты времени, изменяя яркость свечения настольной лампы или торшера. И настольная лампа, и торшер включаются в розетку, имеющую фазовый и нулевой провод. Организовать сканирование силового напряжения для выявления переходов через ноль не сложно, получатся разные решения, зависящие от конкретных условий реализации.

А вот при использовании модуля вместо обычного выключателя мы столкнемся с трудностью – отсутствием нулевого провода в зоне выключателя. Без него мы не можем синхронизировать подачу управляющих сигналов на триак с моментами перехода напряжения через ноль. Если бы нулевой провод был в зоне выключателя, для получения синхронизирующих импульсов можно было бы использовать другой тип оптрона: светодиод – фототранзистор. Светодиод через конденсатор (чтобы не устанавливать резистор большой мощности) и резистор (ограничивающий ток через светодиод) мы подключили

бы между фазой и нулем силового напряжения. При обычном светодиоде (не двухполярном) мы получили бы возможность отмечать один из двух переходов через ноль за период силового напряжения. Но как быть, если нулевой провод отсутствует? Можно поступить двояко – в том месте, где будет установлен системный блок питания 12 В, соединить общий схемный провод с защитным заземлением. При этом в точке установки модуля на место выключателя мы будем иметь возможность сделать схему синхронизации. Второй вариант – подключение цепи оптрона для синхронизации управления параллельно триаку. Напряжение на нем выглядит, как показано на рис. 2.8 или на рис. 2.9.

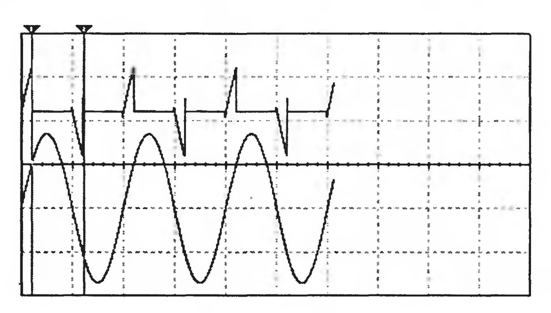


Рис. 2.8. Осциллограмма напряжения на триаке при большой яркости

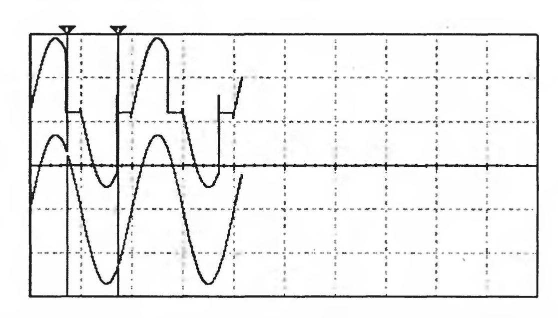


Рис. 2.9. Осциллограмма напряжения на триаке при маленькой яркости

Как видно из рисунков, мы можем осуществить синхронизацию.

Определившись со схемой регулировки, постараемся решить, как будем регулировать яркость – плавно или ступенями.

В любом случае мы будем регулировать яркость ступенями, но их можно сделать маленькими или большими. На мой взгляд, градация переходов с тремя-четырьмя уровнями яркости в полной мере устроит любого. А большинству пользователей достаточно будет двух ступеней – полная яркость и треть яркости. Вопрос этот не принципиальный, и каждый решит его сам.

Второй аспект данной проблемы – использовать ли для временной задержки подачи управляющего импульса встроенный таймер или пустой цикл в программе? Или реализовать эту процедуру с использованием встроенного в микроконтроллер PIC16F628A блока ШИМ. ШИМ – аббревиатура названия способа Широтно-Импульсной Модуляции. Я остановлюсь на максимально наглядном варианте – использую пустой цикл.

Наконец, последнее, что меня в настоящий момент может беспокоить, – не будет ли свет «мигать»? Причина этого беспокойства в том, что контроллер будет осуществлять несколько процессов:

- сканировать напряжение сети ~220 В для определения моментов перехода напряжения через ноль, нужных для синхронизации;
- запускать управляющий импульс, который включит триак, по истечении времени задержки после обнаружения перехода через ноль напряжения сети;
- прослушивать порт USART с целью выявления команд, адресованных модулю.

Системные команды следуют одна за другой. Модуль будет реагировать на все команды, даже когда они адресованы не ему. Пока модуль разбирается с системными командами, он может пропускать управление триаком, а светильник будет выключаться.

Попробуем «прикинуть», как это будет выглядеть. Длительность полуволны частой 50 Гц составляет 10 мс. Возьмем средний интервал «свободного» от управления времени 5 мс. С другой стороны при скорости в сети 2400 бит/с прием одного байта займет около 4 мс. Остается надеяться, что я неправильно понимаю работу USART. Проверим. Если не получится, я готов отказаться от регулировки яркости. Но тем,

кому очень хочется иметь модуль выключателя света с регулируемой яркостью, пока можно предложить:

- поэкспериментировать с изменением тактовой частоты контроллера до 20 МГц и скоростью сетевой работы 115 200 бит/с;
- поэкспериментировать с работой по прерыванию (для сетевых команд);
- в основной программе все команды и запросы перемежать паузами, что замедлит сетевую работу системы, но может оказаться не столь ощутимо, в конечном счете;
- если совсем ничего не получится, использовать внешнюю микросхему обслуживания управления включения света, например контроллер для связи с системой, а полученное время задержки переписывать во внешний счетчик, который будет синхронизирован с частотой сети 50 Гц, и подавать управляющие импульсы на оптрон;
- поискать, нет ли еще каких-либо интересных решений.

Тем не менее, попробуем реализовать программу и проверить ее работу в MPLAB.

Требования к модулю диммера:

- регулирование яркости лампы накаливания на 3 уровнях: полная яркость, половинная яркость, минимальная яркость;
- получение команд от центрального управляющего устройства в формате L14$1N, где 1 – полная яркость, 5 – выключение;
- статус модуль не передает.

Для тех, кто хотел бы запрашивать состояние диммера, я думаю, не составит труда взять эту часть программы из предыдущих решений. Аналогично обстоит дело и с уровнями яркости, командой выключения и полного включения.

Реализация модуля. Вывод RA3 подключим к сканирующему оптрону. Переход через ноль будет отображаться состоянием «1». Для основного программного модуля контроллера два события – появление системной команды и импульс синхронизации с силовой сетью – будут служить точками отклика.

Переделав текст программы, немного «повозившись» с устранением остаточных ошибок после переделки, я вначале сталкиваюсь с проблемой (в отладчике MPLAB) плохой обработки сканирования силовой сети. Справившись с этим, я никак не могу получить изменение яркости (предполагаемой) из-за того, что сетевые команды «пропадают». В конечном счете, программа приобретает следующий вид.

Программа регулировки яркости на языке С

Файл заголовка

```
void putch(unsigned char);
unsigned char getch(void);
int init_comms();
int sim_num_adr();
int cmd();
```

Основной файл

```
#include <pic16f62xa.h>
#include <stdio.h>
#include "dimmer.h"

unsigned char input;      // Для считывания приемного регистра.
unsigned char MOD_SIM1;   // Первый символ адреса модуля.
unsigned char MOD_SIM2;   // Второй символ адреса модуля.
int LEVEL = 100;      // Уровень яркости.
unsigned char command_reciev [6]; // Массив для полученной
команды.
unsigned char COMMAND;    // Флаг прихода сетевой команды.
int MOD_ADDR;      // Заданный адрес модуля, как число.
int sim_end_num = 0;      // Полный символьный номер модуля.
int MOD_NUM;       // Полученный адрес модуля, как число
int i;
int k;
              // Получение байта.
unsigned char getch()
{
   while(!RCIF)    // Когда регистр не пуст.
      continue;
   return RCREG;
}
```

```
            // Вывод одного байта.
void putch(unsigned char byte)
{
   while(!TXIF)    // Когда регистр пуст.
      continue;
   TXREG = byte;
}

// Преобразуем символьный адрес в число.
int sim_num_adr()
{
      sim_end_num = 0;
      MOD_SIM1 = command_reciev [1]; // Первый символ номера.
      MOD_SIM2 = command_reciev [2]; // Второй символ номера.
      MOD_SIM1 = MOD_SIM1 - 0x30;
      MOD_SIM2 = MOD_SIM2 - 0x30;
      sim_end_num = MOD_SIM1*0x0A + MOD_SIM2;
      return sim_end_num;
}

// Выполнение команды.
int cmd()
{
      command_reciev [0] = "L";
      for (i=1; i<3; ++i)       // Запишем номер модуля.
      {
         input = getch();
         command_reciev [i] = input;
      }

      MOD_NUM = sim_num_adr(); //  Полученные номер, как число.
      if (MOD_NUM == MOD_ADDR)  // Если номер модуля совпадает.
   {
      MOD_NUM = 0;
      for (i=3; i<6; ++i)      // Запишем команду в массив.
      {
         input = getch();
         command_reciev [i] = input;
      }
      command_reciev [4] = "3";    // Для получения иллюстрации.
      LEVEL = (command_reciev [4] - 0x30) * 0x64; //Уровень
яркости.
      for (i=0; i<6; ++i) command_reciev [i] = " ";
```

```
        COMMAND = 0;            // Сбросим флаг команды.
    }
    else
    {
        for (i=0; i<6; ++i) command_reciev [i] = " ";//Очистим
массив.
        COMMAND = 0;            // Сбросим флаг.
    }
}

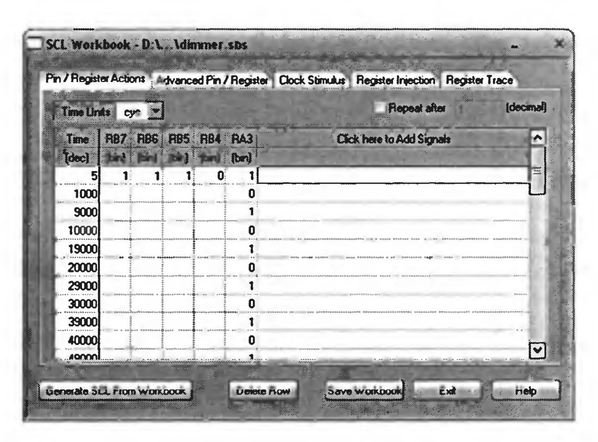
int init_comms()        // Инициализация модуля.
{
PORTA = 0;          // Настройка портов A и B.
CMCON = 0x7;
TRISA = 0b00001000;
TRISB = 0xFE;
RCSTA = 0b10010000;         // Настройка приемника.
TXSTA = 0b00000110;         // Настройка передатчика.
SPBRG = 0x68;           // Настройка режима приема-передачи.
RB0 = 0;            // Выключаем драйвер RS485 на передачу.
CREN =1;
RA0 = 1;
}

void main(void)
{
// Инициализация модуля.
    init_comms();
    for (i=0; i<6; ++i) command_reciev [i] = " ";
    COMMAND = 0;
// Прочитаем и преобразуем номер модуля.
    MOD_ADDR = PORTB;           // Номер модуля в старших битах.
    MOD_ADDR = MOD_ADDR>>4;         // Сдвинем на четыре бита.
// Начинаем работать.
start:
        if ((RA3 == 1)&(COMMAND == 0)) // Сканирование ~220 В,
если нет
        {               //  сетевой команды.
            for (k=0; k<LEVEL; k++) RA1 = 0;
            RA1 = 1;
            if (RCIF)    // Если пришла команда по сети.
            {
                COMMAND = 1; // Устанавливаем флаг команды.
```

```
            input = getch();
            if (input == "L") cmd(); // Если наш модуль.
            else COMMAND = 0;  // Иначе сбросим флаг.
        }
    }
  goto start;
}
```

Выделенная строка добавлена только для получения иллюстрации, как это описано ниже.

Вначале приведу таблицу настроек (рис. 2.10).



Time (dec)	RB7 (bin)	RB6 (bin)	RB5 (bin)	RB4 (bin)	RA3 (bin)
5	1	1	1	0	1
1000					0
9000					1
10000					0
19000					1
20000					0
29000					1
30000					0
39000					1
40000					0
49000					1

Рис. 2.10. Таблица настроек стимулов

Эти настройки пришлось продолжить до времени 140 000. При наладке я выбрал единицы измерения – циклы.

Этим не ограничилось – пришлось изменить частоту с 4 МГц на 20 МГц, изменить основной текст в функции приема команды, закомментировав часть текста:

```
int cmd()
{
/*     command_reciev [0] = "L";
       for (i=1; i<3; ++i)       // Запишем номер модуля.
       {
           input = getch();
           command_reciev [i] = input;
```

```
      }

      MOD_NUM = sim_num_adr(); // Чтение из сети.

   if (MOD_NUM == MOD_ADDR)
   {
      MOD_NUM = 0;
      for (i=3; i<6; ++i)      // Запишем команду в массив.
      {
         input = getch();
         command_reciev [i] = input;
      }  */
      command_reciev [4] = "3";
      LEVEL = (command_reciev [4] - 0x30) * 0x64;
      for (i=0; i<6; ++i) command_reciev [i] = " ";
      COMMAND = 0;
/* }
   else
   {
      for (i=0; i<6; ++i) command_reciev [i] = " ";
      COMMAND = 0;
   }  */
}
```

Только после всех этих обманных маневров мне удалось получить иллюстрацию к тому, что я хотел сказать. После получения команды интервал от момента перехода напряжения сети ~220 В через ноль, до момента включения триака, отмеченный на рис. 2.11 маркером, меняется на новый интервал, отмеченный на рис. 2.12.

Что можно сказать в данный момент? Отработка сканирования и включения выглядит нормально (рис. 2.13 и 2.14).

Уровень яркости определяется переменной LEVEL, которая изменяется командой, получаемой от компьютера. На этих рисунках видно, что импульсы управления идут с частотой 100 Гц (10000 мкс). Вышеприведенные картинки, полученные «жульническим путем», дают надежду на то, что команду от компьютера все-таки удастся прочитать. Мои опасения по поводу «мигания» света, скорее всего, беспочвенны. Их лучше отнести к опасениям по поводу отработки сетевых команд.

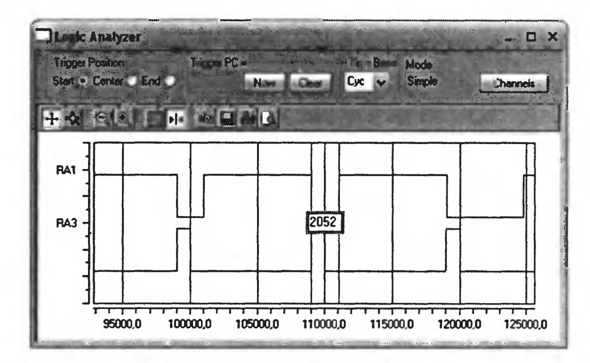


Рис. 2.11. Отладка управления триаком

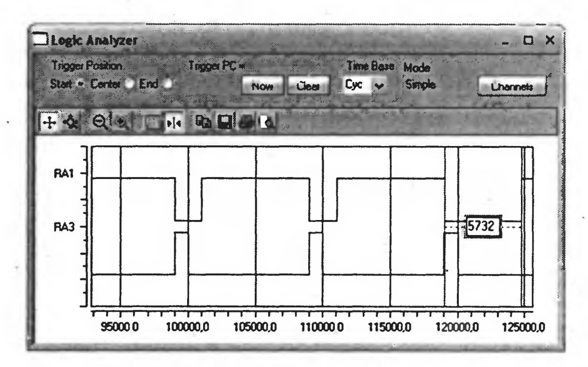


Рис. 2.12. Изменение управления триаком

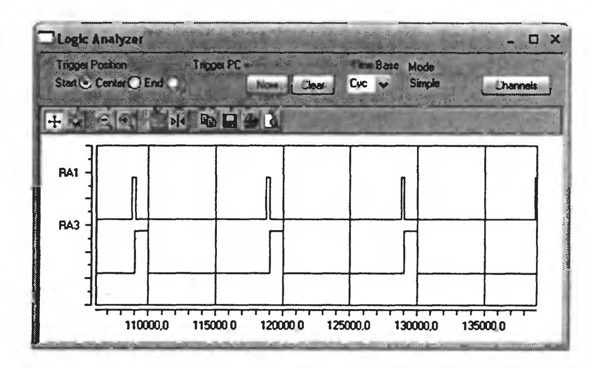


Рис. 2.13. Отладка изменения яркости

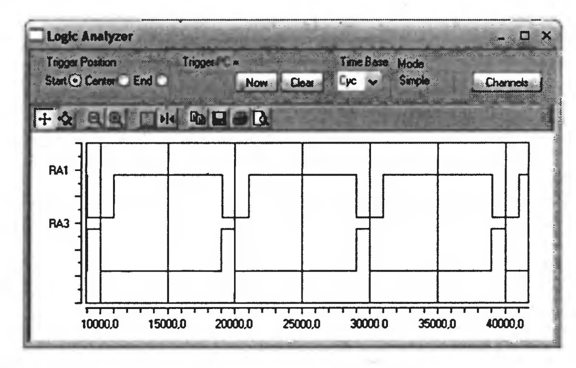


Рис. 2.14. Другое значение яркости

Для очистки совести мне следовало проверить работу модуля на макете. Но я не хочу работать с силовой сетью ~220 В.

Завершая разговор о диммере, приведу HEX-файл модуля управления яркостью.

HEX-файл модуля управления яркостью

```
:1000000830100308A000428203084003530162017
:10001000830103330B800FE30B7003530840037303C
:100020001A2083012E2B04068001840A0406031D76
:1000300013280034F00026208000840A040870068B
:1000400031900341B2883120313B700B81B31288F
:10005000381B392838088A003708B70A0319B80A44
:1000600082008313381883173708B70A8400000802
:020070000080086
:1005B20083018C1ED92A1A0808008301B400831211
:1005C20003130C1EE02A340899000800F401F50117
:1005D2000310F30CF20C031CF52A7008F4077108DF
:1005E2000318710AF5070310F00DF10D7208730478
:1005F20003190034E92A8301850107309F0008307E
:1006020083168500FE3086009030831298000630F3
:100612008316980068309900831206101816051484
:1006220008008301AC01AD012F08A1003008A2002F
:10063200D030A107A2070A30F200F3012108F0002E
:10064200F101E72222087407AC0075080318750A45
:10065200AD00F1002C08F0000800FC22A801A9015D
:100662003B2B28082E3E8400831320308000A80AEA
:100672000319A90A2908803AF00080307002063076
```

```
:1006820003192802031C322BA0010608A400A501AD
:10069200430F000250DA50CA40CF00B4B2B83129B
:1006A2000313851D502BA008031D522BAA01AB0179
:1006B2002B08803AF0003608803A7002031D632B43
:1006C20035082A02831203130318 6C2B8510AA0A19
:1006D2000319AB0A592B85148C1E502BA001A00ABA
:1006E200D922A3004C3A031D782B7A23502BA00168
:1006F200502B4C308301AE00A801A80AA9018C2B13
:10070200D922A30028082E3E84008313230880 00E8
:10071200A80A0319A90A2908803AF0008030700259
:100722000330031928020 31C812B12237008A60030
:100732007108A7002506031DA02B2408260 6031D09
:10074200E62BA601A7010330A800A9012908803AD7
:10075200F00080307002063003192802 0318BD2B06
:10076200D922A30028082E3E840083132308800088
:10077200A80A0319A90AA72B6430F200F301320870
:10078200F000F101D030F0070318F10AFF30F10751
:10079200E7227408B5007508B600A801A901290866
:1007A200803AF00080307002063003192802 0318E4
:1007B200E42B28082E3E84008313203080 00A80AF0
:1007C2000319A90AD02BA0010800A801A901290830
:1007D200803AF00080307002063003192802 0318B4
:1007E200FC2B28082E3E840083132030800 0A80AA8
:0E07F2000319A90AE82BA001080064340034A2
:00000001FF
```

Файл предназначен для экспериментальных целей. Добавлю еще несколько слов:

- если эксперименты с диммером не приведут к успеху, можно рассмотреть вариант применения более мощного микроконтроллера;
- использовать решения, о которых я говорил выше;
- применить, что проще, два контроллера, один из которых будет общаться с системной сетью, а второй управлять триаком;
- и не забудьте: если вы решите сделать выключатель с регулируемой яркостью, (что никак не отражено в реализации) *следует предусмотреть одну или две кнопки для ручного управления!*
- в режиме ручного управления можно сделать следующее – одна из кнопок при кратковременном нажатии

полностью включает свет, при удержании медленно увеличивает яркость; вторая кнопка проделывает то же самое, но с выключением; думаю, вам не составит труда изменить программу для поддержки этих добавлений.

А я продолжу рассказ о модулях, которые могут войти в систему.

Модуль последовательного интерфейса

Некоторые устройства позволяют управлять ими по интерфейсу RS232, что предпочтительней ИК-управления, если известны команды управления устройством. Это может быть проигрыватель CD-дисков, проектор или видеокамера наблюдения. Модуль позволяет пересылать команды по системной сети к устройству с преобразованием их к интерфейсу RS232.

В данном случае, я думаю, можно применить конвертер RS485–RS232.

Если есть желание расширить возможности модуля, можно сделать модуль с несколькими выходами RS232, добавив контроллер и мультиплексор, переключающий порты RS232. Это только идеи, а не решения. Решения, я думаю, вам интереснее найти самим.

Далее в книге пойдет речь о смешанных системах. При объединении устройств разных систем в единую сеть модуль последовательного интерфейса может избавить вас от поиска сложных решений.

Модуль аудиокоммутатора

Еще один модуль, который может найти применение в системе, это модуль аудиокоммутатора. В принципе, достаточно коммутации громкоговорителей с помощью релейного модуля. Но можно коммутировать линейный выход, например CD-проигрывателя и радиоприемника, с аудиовходом телевизора. Для этой цели, используя релейный модуль, можно вместо реле включать соответствующие электронные ключи. Дешевле всего использовать ключи цифровой 561 серии. Микросхема

К561ТК3 (функциональный аналог 4066) вполне успешно работает в качестве управляемого ключа.

Микросхема допускает двухполярное питающее напряжение, при этом она передает низкие частоты от «0». Но микросхема допускает и однополярное включение питающего напряжения. При этом сигнал лучше передавать через конденсатор соответствующей емкости. При реализации не следует забывать, что коммутатор должен позволять подключение к одному выходу источника сигнала нескольких входов приемников, но не наоборот. Если вам понадобится сделать микшер (смеситель сигналов), несколько выходов следует подключать к одному входу через развязывающие резисторы.

Для аудиокоммутатора важно проверить, не ухудшается ли в значительной мере соотношение сигнал/шум. Конечно, я не имею в виду применение подобного модуля для аппаратуры класса Hi-End.

Модуль видеокоммутатора

Еще более интересными мне представляются эксперименты по превращению модуля аудиокоммутатора в видеокоммутатор. Я не уверен, что это осуществимо, но попробовать стоит, используя новые цифровые микросхемы серии 1561 или 1564. Собственно, видеокоммутатор в отношении логики ничем не разнится с аудиокоммутатором. Необходимо только согласование нагрузки. Видеоцепи работают на сопротивление 75 Ом, то есть, подразумевается коаксиальный кабель с волновым сопротивлением 75 Ом, и разъемы желательно применять соответствующие. Коммутатор должен быть согласован. Этого можно достигнуть, применяя входные и выходные усилители на микросхемах, хорошо работающих с видеосигналом (я применял, когда возникла необходимость, микросхемы AD8055AN). Его второе отличие от аудио коммутатора – полоса частот. Для видеокоммутатора она должна быть не менее 0–6 МГц при композитном сигнале.

Модуль управляемого усилителя

На основе микросхем усилителей в сочетании с управляющим контроллером, добавив реле для подключения питающего напряжения к усилителям и схемы ключей на микросхемах К561ТК3 (или аналогичных электронных ключей), можно реализовать модуль управляемого усилителя (в частности, многоканального). Реле будет включать и выключать питание усилителя по команде компьютера. Ключи будут коммутировать резистивный делитель для ступенчатой регулировки громкости по командам компьютера или системного устройства управления. А управляющий контроллер, вероятно, будет мало отличаться от примененного в релейном модуле.

С введением в систему управляемого усилителя расширяются возможности отображения информации. Когда я говорил о применении уличного термометра в системе, то предлагал «помигать светом», если температура за окном ниже, чем –15°. Гораздо приятнее получить сообщение от системы в виде голосового напоминания: «Не забудьте одеться теплее! На улице мороз!».

Голосовые системные сообщения можно организовать на компьютере, особенно, если вы используете компьютер для управляющей программы. В качестве альтернативы – неисправный телефонный аппарат с цифровым автоответчиком. Сам телефонный аппарат может быть неисправен, а автоответчик вполне работоспособен. Сегодня, порой, выбрасывают и вполне исправные телефонные аппараты, заменяя их аппаратами, позволяющими подключить три-четыре трубки. А телефонные аппараты с цифровым автоответчиком лет десять назад стоили не дороже обычных.

Можно собрать и самому цифровой модуль для записи и воспроизведения голосовых сообщений. Интересно было бы использовать микроконтроллер, АЦП и внешнюю память для сохранения оцифрованных сообщений. Но вернемся к усилителям.

Сейчас можно недорого приобрести микросхемы вполне качественных усилителей небольшой (по сегодняшним меркам) мощности. С мощностью усилителей вопрос особый. Необходимая для создания нормальной звуковой картинки мощность очень сильно зависит от применяемых громкоговорителей. Вернее от их фактического КПД. Тяга к усилителям большой мощности возникла, в первую очередь, из-за желания передать низшие частоты звукового диапазона. Чем ниже желаемая частота, тем труднее ее воспроизвести, что требует большей мощности от громкоговорителя и усилителя. Но не следует забывать, что в реальном помещении небольшой площади без специальной и достаточно сложной (и дорогой) обработки помещения воспроизвести частоты ниже 40–50 Гц слишком сложно. А если ограничиться нижней частотой воспроизведения 40–50 Гц, при правильной конструкции акустического оформления необходимая мощность усилителя может не превышать 5–10 Вт.

Но это тема совсем другого разговора.

В отношении системы мне хотелось бы добавить некоторые соображения. Давайте, подумаем, в каких случаях нам требуется звук в помещениях без источников звука:

- есть большие любители послушать любимую музыку в ванной;
- приятно, когда к вам приходят гости, включить музыку в прихожей или гостиной, чтобы в ожидании обеда под музыку пить коктейли;
- в столовой, где собрались гости, можно включить музыку, которая слегка заглушит стук ножей и вилок и позволит соседям по столу переговорить без того, чтобы привлекать внимание всего стола;
- наконец, после обеда можно и потанцевать в холле.

Я привел несколько примеров применения аудиодистрибуции в реальной системе умного дома. Какой в данных случаях разумно реализовать вариант физического построения аудиоподсистем. Во всех выше перечисленных случаях лучше использовать встроенные потолочные громкоговорители. И, я почти уверен, работающие в режиме «моно». Получить от стереосистемы с потолочными громкоговорителями

качественную звуковую картину, мне кажется, слишком сложно, да еще и на большой площади!

Нужно ли заботиться о большом динамическом диапазоне звука? А какой в этом прок, если музыка должна оставаться «мягким фоном». То есть, большая мощность не нужна.

Нужно ли передавать весь звуковой частотный спектр? Не думаю. Скорее следует позаботиться о том, чтобы, ограничив частотный спектр снизу частотой 60–70 Гц, не забыть ограничить его сверху частотой 10–12 кГц, учтя при этом особенность восприятия звука при небольшой громкости и внеся частотную коррекцию. Правильно построив эту часть устройства звуковоспроизведения, можно ограничиться уровнем громкости, имеющим одно приемлемое значение, что избавит от необходимости регулировать громкость.

Словом, не следует переоценивать возможности «звука сверху», затрачивая силы на получение очень хороших параметров звучания, если они никогда не будут применяться. Лучше позаботиться о правильном построении всего тракта и подобрать музыку, которая была бы приятна гостям, не отвлекала их ни от беседы, ни от вкусной еды за столом.

И, например, если вы намерены озвучить свою ванную комнату, не пытайтесь получить нечто небывалое, используя самое высококачественное оборудование. Условия воспроизведения звука не позволят реализовать все эти преимущества.

Высококачественное оборудование требует специального помещения с хорошей акустической обработкой. Если вы можете себе позволить подобное помещение было бы хорошо, если б специалисты из лаборатории акустики проверили помещение с помощью своих приборов. Это позволит внести коррекцию в оборудование с целью получения максимального качества звука в данном конкретном случае.

Модуль системного ИК-пульта управления

На базе решений для модуля цифровых вводов и модуля управляющих ИК-кодов можно разработать системный пульт с небольшим количеством команд. В качестве излучающего

светодиода можно использовать светодиоды от старых пультов управления (или купить аналогичный) или любой светодиод ИК-диапазона (либо захватывающего ИК-диапазон). В последнем случае их следует включать короткими импульсами с током, превышающим средний допустимый ток, но с большой скважностью. Так, кстати, работают и промышленные пульты управления (если не все, то некоторые).

Конечно, возможности такого пульта будут ограничены, как в количестве управляющих клавиш, так и в количестве системных кодов. Но о том, как можно увеличить количество клавиш для управления, мы говорили в разделе, относящемся к модулю цифровых вводов. А увеличение количества системных кодов зависит от реализации. Взяв за основу простейшие ИК-коды, можно создать функцию воспроизведения такого кода, где параметр, меняющий код, может иметь значение, умещающееся в одном байте. В этом случае энергонезависимая часть контроллера уместит достаточно много кодов. Я не знаю готового решения, но полагаю, вам будет интересно попробовать реализовать этот вариант модуля в составе системы. Подразумевается, что пульт управления будет переносным, а в этом случае не забудьте, что контроллер предлагает энергосберегающий режим управления.

Модуль аналогового ввода для термометра

Модули аналоговых вводов, как мне кажется, чаще требуются в профессиональной деятельности, в технологических процессах. Там, где есть необходимость считывать показания датчиков, отображающих непрерывно меняющиеся параметры в широком диапазоне.

Но я не исключаю интерес со стороны любителей к созданию подобных устройств. С моей точки зрения, системное устройство лучше сделать в виде самостоятельного модуля целевого назначения, что избавит от необходимости прокладывать экранированные провода. И только после того как продумано его применение в системе. Даже достаточно простая задача отображения непрерывно меняющейся

информации, например температуры, может быть проще решена с помощью специализированного АЦП, соединенного с дисплеем. Как это сделано в мультиметре. Получается простая, надежная и многофункциональная конструкция.

Это, конечно, не означает, что я исключаю возможность построения датчика полностью на микроконтроллере. Это могло бы быть интересной задачей.

Замена проводного канала RS485

Для тех, кто, проведя первые эксперименты с системой, пожелал бы использовать дома что-то из разработок, препятствием к реализации задуманного может стать отсутствие в доме развитой кабельной системы, необходимой для организации сети. Прокладывать провода поверх плинтуса или по стенам? Можно, проведя предварительные эксперименты, попробовать заменить провода радиоканалом. В качестве приемопередатчиков можно использовать готовые модули, используемые при построении управляемых авиамоделей и моделей автомобилей. В продаже есть и одноканальные, и многоканальные радио модули для радиоуправляемых моделей. Они настроены на разрешенные для этой цели радиочастоты, невелики по габаритам и, я думаю, удобны в применении. Заменив микросхему интерфейса RS485 на подобные модули, можно попытаться обойтись без проводов. Сам я не пробовал, но желание купить модули и испытать их в работе было.

При отказе от проводной связи потребуется снабдить каждый модуль системы собственным блоком питания, но решение этой проблемы может оказаться более простым, чем прокладывание проводов по комнате или по квартире. Хотя и с проводами – в настоящее время есть плоские кабели очень небольшой толщины. Такие кабели можно аккуратно разместить под обоями возле пола, и они не испортят вида комнаты.

Другим решением этой проблемы может стать использование ИК-канала. При этом для ретрансляции сигнала между комнатами потребуются специальные модули ретрансляции. Их легко сделать на базе модулей ИК-приемников и ИК-излучателей. К недостаткам решения следует отнести необходимость в большом количестве ретрансляторов. Но в продаже

есть ИК-зеркала, которые можно использовать в качестве ретрансляторов ИК-сигналов. Словом, здесь тоже большой простор для экспериментов.

Усовершенствование базовых модулей

Следует отметить, что все описанные в книге модули далеки от совершенства. Хотя большая их часть была мной проверена на макете, я не проверял их совместную работу. Подобная проверка может открыть большой простор для творчества и по устранению недостатков, и по усовершенствованию самих устройств. При этом усовершенствование может касаться как увеличения функциональности, так и надежности. Сама микросхема контроллера настолько надежна, что хотя бы из уважения к этому факту следует не оставлять без внимания вопрос надежности устройства в целом.

Одним из усовершенствований, о которых я упоминал, может стать использование встроенной памяти EEPROM для задания адреса модуля. В промышленных разработках адрес устройства задается программно. Для этой цели один из вводов устройства используется для перехода к режиму настроек. Когда он соединен с общим проводом, при включении питающего напряжения модуль переходит в специальный режим настроек. По сети ему передаются параметры – адрес модуля, скорость сетевой работы, – которые модуль запоминает в энергонезависимой памяти. Если применить построение модуля с внешним кварцем на 20 МГц, диапазон сетевых скоростей расширяется до 115200 бит/с. Сами по себе эксперименты с разной скоростью сетевой работы могут оказаться не менее увлекательными, чем разработка модулей. В этом случае возможность программного изменения скорости работы USART очень полезна.

Наконец, микроконтроллер PIC16F628A не является самым мощным в серии PIC-контроллеров. Можно провести эксперименты с контроллерами PIC18Fxxx или другими. Выбор контроллера вне профессиональной разработки может определяться разными факторами: наличием подходящего

контроллера в продаже, доступностью его по цене. Может играть роль и простой интерес к конкретной микросхеме. Главное, что я хотел показать на протяжении книги, что освоение работы с микроконтроллерами доступно любому, кому интересно, а интересно любому, кому нравится творить, созидать, изобретать. Даже если изобретаешь велосипед, это занятие оказывается много занимательнее, чем покупка готового.

Последние замечания

Как аппаратная, так и программная разработка выполнена мною в макетном варианте. Я говорил, что не намерен создавать промышленный вариант – моя задача показать, что есть такое интересное занятие – придумывать и создавать, используя возможности компьютера, интересные устройства. По этой причине компьютер выбран в качестве основного управляющего устройства. Нс, если кому-то захочется, не ограничиваясь экспериментами, использовать систему в своей комнате или квартире, я бы посоветовал рассмотреть возможность замены компьютера на автономное управляющее устройство, выполненное на базе, например, того же микроконтроллера PIC16F628A. Его программная память позволяет разместить до 2 Кб управляющей программы, что вполне может оказаться достаточным для управления модулями в комнате или даже квартире. Можно, если памяти окажется недостаточно, использовать другой микроконтроллер или дополнительную память для размещения программы.

Программу для создания основной программы можно написать в виде, близком к профессиональному, – очень интересная задача. И очень интересно придумывать новые модули, которые были бы полезны в составе системы.

Если добавить в виртуальную лабораторию на компьютере программу, позволяющую провести разработку на традиционной элементной базе – транзисторы, резисторы, конденсаторы и т.д., возможности разработки многократно увеличатся. Необычайная гибкость микроконтроллеров, надеюсь, мне удалось это показать, позволяет за счет небольших изменений в программе превратить микроконтроллер в основу совсем

другого устройства. Даже замена элементов, которыми управляет контроллер, без замены его программы меняет конечное назначение устройства – замена реле на управляемые ключи превращает релейный модуль в управляемый аудиокоммутатор или усилитель. И всю эту работу можно осуществить за компьютером, отложив покупку деталей до того момента, когда вы уверитесь в необходимости создания прототипа, и реальности достижения цели.

Несколько практических советов для тех, кто намерен не останавливаться на экспериментах, а воплотить систему в жизнь. Как все системы подобного назначения, эта получается достаточно гибкой. Она может использоваться в комнате и во всей квартире, как система, управляемая с компьютера или как децентрализованная система. Достаточно немного изменить программы модулей. Например, модуль цифровых вводов, используемый в качестве контроллера клавиатуры, может отправлять прямые команды модулям выключателей света (или релейным). В итоге вы получаете автономную подсистему – в одном месте расположен пульт управления всеми выключателями в квартире (или в каждой комнате находятся пульты управления всеми выключателями света в квартире), а исполняющие модули расположены рядом со светильниками. Подобным же образом можно организовать любую подсистему.

Для опытов со светом я бы посоветовал использовать настольные лампы, бра, торшеры, включаемые в розетки; задействовать для их подключения промежуточную розетку, подключаемую через контакты реле (релейного модуля) или триак (модуля выключателя света). Включать промежуточную розетку в сеть следует только после тщательной проверки правильности и качества монтажа.

Для тех, кто все-таки решит заменить штатный выключатель света модулем и не забудет на время работ по замене отключить автомат в силовом шкафу, хочу напомнить, что кроме управления выключателем по системной сети следует предусмотреть ручное выключение. Например, изменив программу контроллера, – установить один или два вывода порта на ввод. К этим выводам будет подключаться кнопка (или кнопки) ручного управления. Триак лучше установить на теплоотвод в виде

металлической пластины, которая одновременно может играть роль несущего элемента конструкции. Выключатель для этих целей можно подобрать удобный к переделке. Я не исключаю, что можно расположить все элементы схемы модуля внутри любого обычного выключателя и использовать его контракты для ручного управления модулем.

Если вы захотите использовать датчик температуры для управления калорифером, не забудьте, что используемый вами термозависимый элемент может подвергаться воздействию случайных движений воздуха. Если тепловые процессы в помещении инерционны, сам термоэлемент может обладать значительно меньшей инерционностью. Используйте либо разнесение по температуре команд включения и выключения обогревателя, либо используйте временные задержки, проверяя состояние датчика температуры дважды с некоторым интервалом времени.

И последнее: я выбрал в качестве иллюстрации систему «Умный дом», но это не единственная система, которую можно реализовать с использованием микроконтроллера. Придумывать подобные системы и создавать их, используя сегодняшние возможности компьютера, – увлекательнейшее занятие, даже если не превращать его в профессию.



То, что рядом с «Умным домом»

Предположим, что вы человек практичный. Вам не достает только экспериментов и того удовольствия, которое они могут принести. Дочитав описание, вы решили реализовать систему. В том, что касается микроконтроллера, вам все понятно. Но, выбрав концепцию (одну из предложенных в Приложении) и внеся изменения в программу модуля диммера, вы решили собрать его. Если вы, в отличие от меня, специализируетесь на тиристорных схемах управления, у вас не возникнет проблем. Но я, прежде чем начать «жечь» триаки и оптроны, постараюсь проверить, а прав ли в том, как нарисовал схему? Для этой цели я воспользуюсь одной из программ, о которых пойдет речь ниже.

Вы не хотите устанавливать ИК-излучатель возле телевизора для включения его из системы. Вам удобнее сделать мощный излучатель, который расположится в другом конце комнаты. Даже воспользовавшись моим советом добавить конденсатор параллельно резистору (об этом тоже упомянуто в Приложении), вы должны определиться с величиной емкости. Ее можно рассчитать, используя постоянную времени, но помните ли вы, как это следует сделать?

Положим, прочитав о модуле аудиокоммутатора, вы подумали, что его можно применить не только для распределения музыки по квартире, с чем вполне справлялся релейный модуль, но и для устройства интеркома. Квартира большая, кричать на всю квартиру – не накричишься. Берем электретный

микрофон, добавляем усилитель, небольшие громкоговорители от компьютера во всех комнатах и получаем с помощью коммутатора вполне работающую систему интеркома. Осталось понять, какой нужен усилитель. Первое, что мне приходит в голову, – применить операционной усилитель. Но я тут же вспоминаю, что он потребует двухполярного питания. Есть, безусловно, схемные решения с однополярным питанием, но я не уверен, что единственный транзистор не справится с этим лучше. Можно найти подходящую к случаю микросхему. Но я могу засомневаться, что так будет лучше, вспоминая, как много лет назад пытался решить схемный вариант включения источника с промежуточным выходным сопротивлением. Проблема была в минимизации шумов. Одни микросхемы прекрасно работали с высокоомными источниками, другие – с низкоомными. А в промежутке между ними – поиск. Вне всяких сомнений, можно решить любую задачу опытным путем, но лучше вначале опробовать решение за компьютером.

Задумавшись о создании интеркома, вы подумали, что жене легче будет ухаживать за вашим маленьким ребенком, если в его комнате поставить микрофон и подключить его к интеркому, установив порог срабатывания так, что интерком будет включаться только тогда, когда ребенок заплачет. Как выбрать порог?

Вам не хочется растягивать провода по квартире. Вы решаете, что оснастите все модули радиоканалом. Правильно. Я посмотрел, что предлагают производители радиомодулей. Есть очень славные образцы. Их, я думаю, можно подключить к микроконтроллеру даже без инвертеров. Но есть у них маленький недостаток – цена (около 200 долларов). Прежде, чем купить с десяток подобных модулей, я бы, при всей моей лени, попробовал собрать что-то подходящее из транзисторов–резисторов–катушек. И проверил бы схему за компьютером.

Решая некоторые задачи, связанные с системой «Умный дом», все равно время от времени сталкиваешься с проблемами, вынуждающими что-то сделать дополнительно, хотя бы только для того, чтобы выиграть время. Прежде чем хвататься за паяльник, я стараюсь максимально тщательно проработать решение.

Один из примеров. Необходимо в подсистеме противопожарной безопасности организовать индикацию состояния датчиков. Есть модуль в единственном экземпляре, имеющий 8 реле. Датчиков 10. Кроме индикации весьма желательно добавить звуковой сигнал тревоги. Если тревога не ложная, система переходит в режим пожарного оповещения, отключает климатическую подсистему и т.д. По первому впечатлению я решаю, что легко могу добиться желаемого с помощью организации матричного подключения светодиодов с матрицей 4×4. Но, поостыв и нарисовав предполагаемую схему в программе Multisim, как показано на рис. 3.1, я прихожу к выводу, что следует позаботиться в программной части о поочередном включении индикаторов. Я не уверен, что обратил бы внимание на эту деталь до предварительного рассмотрения работы схемы.

Среди великолепных специалистов по микропроцессорной технике, программистов, для которых не составит труда привести программу к виду коммерческой версии системы, далеко не редкость столь прочно забыть все, что касается аналоговой схемотехники, и тогда любые расчеты, необходимые

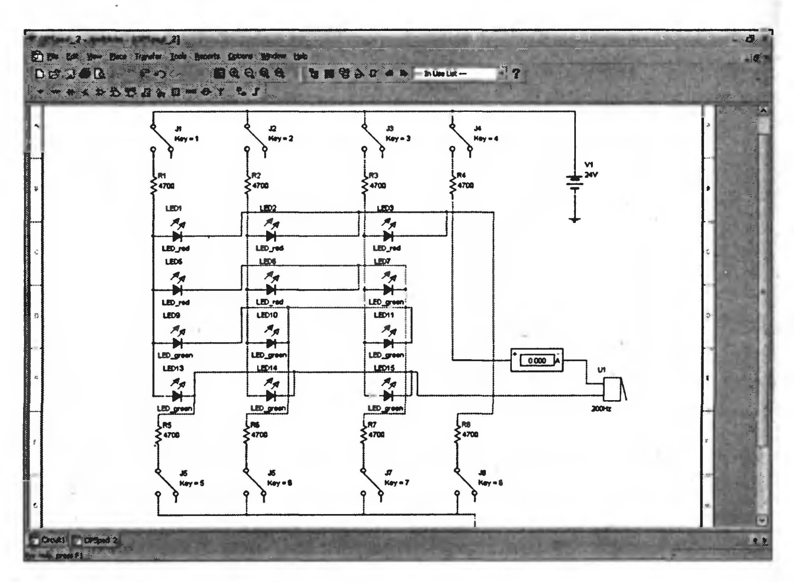


Рис. 3.1. Организация противопожарной индикации с релейным модулем

до макетирования схемы, становятся сущим кошмаром. Что же говорить о любителе? Для всех них программы, помогающие обойтись без расчетов, – подарок судьбы.

Значительная часть оборудования, с которым мне пришлось работать, была закуплена, когда производители не слишком заботились о продвижении на европейский рынок. По этой причине оборудование разрабатывалось с учетом напряжения в силовой сети ~110 В. Привезенное в нашу страну, оно переделывалось специалистами под работу с напряжением ~220 В. Но, я почти в этом уверен, специалисты не располагали всей необходимой для переделки документацией, включая тот неприятный момент, что электрические схемы оборудования отсутствовали. Позже это отсутствие сказывалось на надежности. Время от времени оборудование выходило из строя. Чтобы понять причины этого, приходилось внимательно разбираться с (отсутствующей) схемой. Например, так, как показано на рис. 3.2.

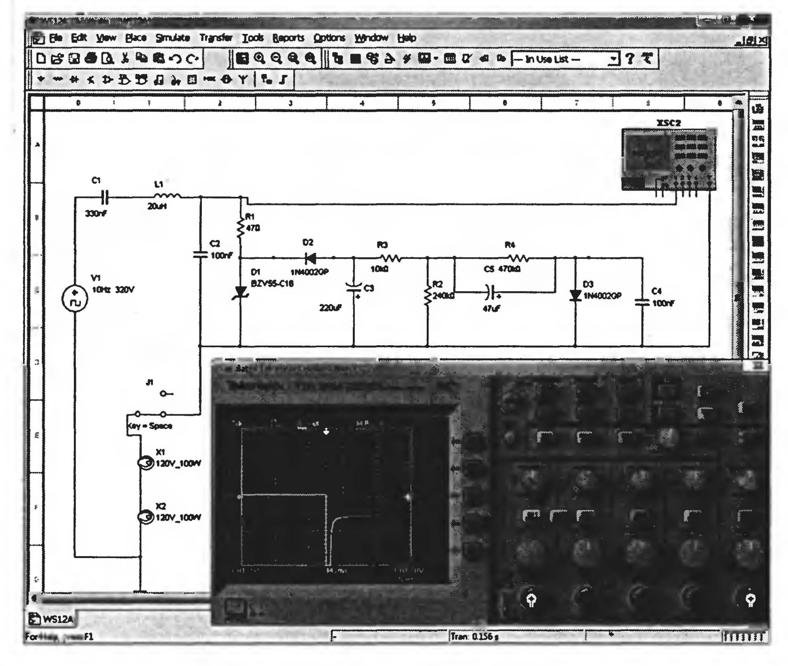


Рис. 3.2. Диммер системы X10 (фрагмент схемы)

За время, которое проходило от момента разработки оборудования, существенно изменялась сфера его применения, компоненты, с которыми оно работало. В результате вполне исправное оборудование, собранное и включенное согласно указаниям, прилагаемым к нему, работало не вполне адекватно ожиданиям. Найти причины подобных явлений мне неоднократно помогало моделирование (и модификация) его частей в программах, о которых я хочу немного рассказать (рис. 3.3).

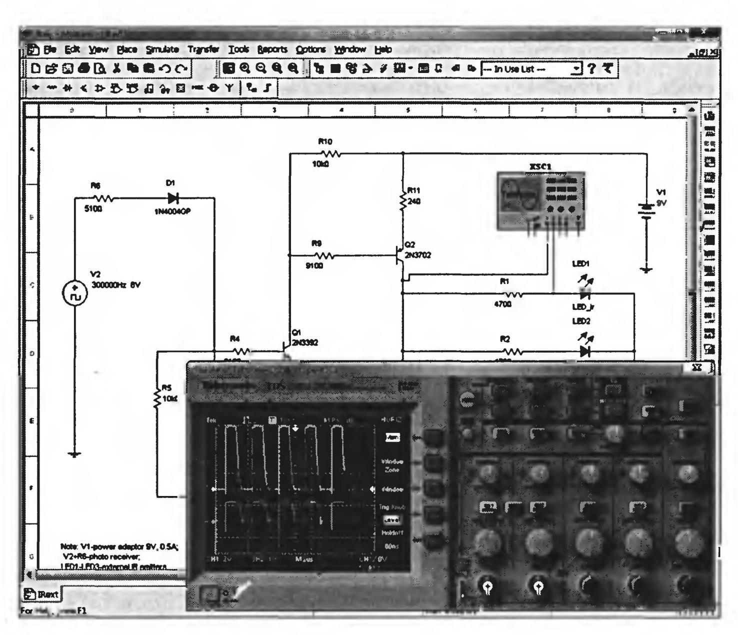


Рис. 3.3. Расширитель ИК-каналов (фрагмент схемы) в программе Multisim 9

MULTISIM

Мне эта программа напоминает макетную плату в сочетании с радиолюбительской лабораторией и большим ящиком, до

верху наполненным радиодеталями. Если не все, то огромное количество схем можно придумать и проверить, не выходя из-за компьютера. Конечно, решающим моментом станет проверка схемы на макетной плате, но значительную часть ошибок (а, может быть, и все) можно найти до покупки деталей. Я не уверен, что работы за компьютером, исключая профессиональную деятельность, достаточно. Но многие неудачи при реализации готовых схем, которые остались «явлением таинственным и темным» в силу того, что плата «заросла» добавленными и замененными элементами, удобно анализировать за компьютером. Осциллограмма, на которой обнаружилась ошибка, выявленная с помощью программы Multisim в достаточно сложной схеме, показана на рис. 3.4.

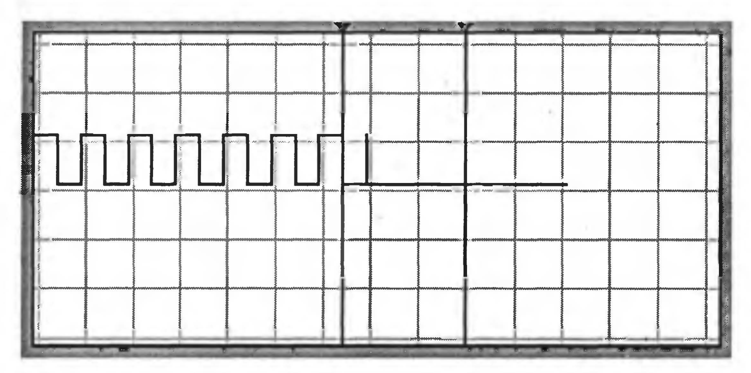


Рис. 3.4. Ошибка, выявленная с помощью программы Multisim

Укороченный импульс, который вы видите на рисунке, в конечном счете, свел бы на нет все усилия, потраченные на разработку, если б не во время проведенный анализ схемы.

После запуска программы на экране появляются стандартные для среды Windows рабочее поле и меню. Раскрывая меню компонентов, в левой части окна можно выбрать необходимые и мышкой перенести их на рабочее поле. В разных разделах меню разные электронные компоненты, индикаторы

и приборы. Для перетаскивания следует подвести курсор к соответствующей иконке меню, щелкнуть левой кнопкой мыши и перенести компонент в поле чертежа.

Выбрав все компоненты, можно разместить их наиболее наглядным образом, используя при необходимости подменю поворота и отражения, которые появляются при щелчке правой кнопки мыши выделенного компонента. При размещении следует иметь в виду, что позже к схеме добавятся приборы, с помощью которых вам можно будет проверить работу схемы, настроить ее, привести к виду, готовому для воплощения, то есть сборки готовой платы.

Кроме того, можно сразу позаботиться о наглядности схемы, придав ей вид, который ВАМ легче понять. Этому моменту, как мне кажется, зачастую совсем не уделяется внимания. Дело в том, что профессиональные чертежи оформляются в соответствии с принятыми нормами и рекомендациями ГОСТов, базирующихся на старых технологиях. В частности, стремятся к наиболее полно заполнить поля чертежа (экономя бумагу), максимально его упростить, чтобы облегчить жизнь чертежникам, которым приходится многократно воспроизводить один и тот же чертеж. В результате чертеж, который легко воспроизводится, трудно читается. Ситуация схожа с книгоизданием, где применение аббревиатуры настолько общепринято, что сделав перерыв в чтении, ты бываешь вынужден вернуться к началу – иначе не вспомнить значение многочисленных АБВГД, ВБЛГ и т.п.

За компьютером вам нет смысла придерживаться этих стандартов, полезнее сделать чертеж легко понимаемым и наглядным, чтобы, возвращаясь к нему, вы могли сразу увидеть схему со всеми ее особенностями (рис. 3.5).

После размещения всех необходимых элементов схемы следует соединить их, используя левую кнопку мыши. Когда курсор находится на краю любого из компонентов, достаточно щелкнуть левой кнопкой, а затем провести соединение к следующему элементу схемы до появления пересечения курсора с линией в точке соединения, где следует повторно щелкнуть левой кнопкой (рис. 3.6).

Рис. 3.5. Вид программы Multisim

Рис. 3.6. Соединение элементов схемы

На схеме можно разместить приборы, выбрав их из меню приборов. Вольтметр и амперметр находятся в меню индикаторов, представленном иконкой семисегментного цифрового индикатора (рис. 3.7).

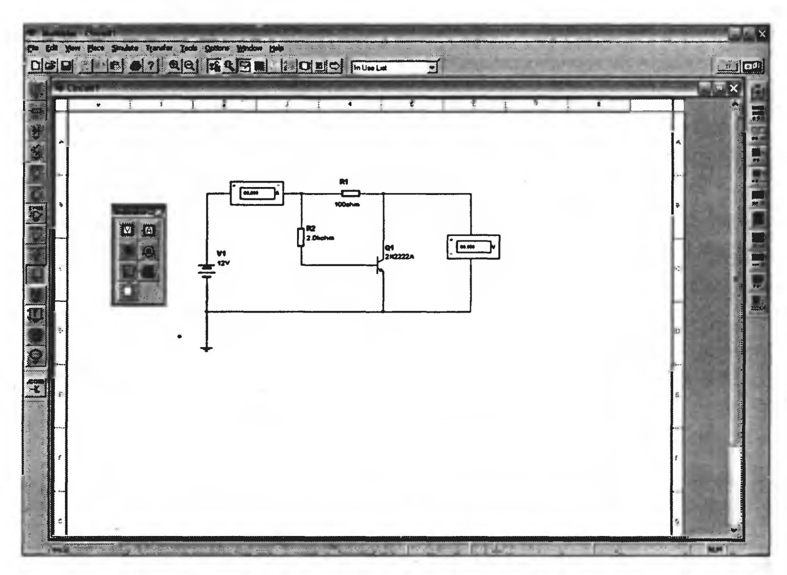


Рис. 3.7. Простые приборы программы Multisim

После включения схемы (в правом верхнем углу есть кнопка включения) приборы покажут измеряемые величины (рис. 3.8).

Если появится необходимость изменить значения параметров компонентов, их следует выделить, дважды щелкнув левой кнопкой мыши. В появившемся диалоговом окне **Component properties** (Свойства компонент) следует выбрать кнопку **Replace** (Заменить) в нижнем левом углу, а затем – новое значение компонента.

Следует отметить, что для работы схемы необходимо использовать заземление, даже если схема не требует его. Проект, который мы осуществим для знакомства с программой, может показаться скучным – усилительный каскад на транзисторе.

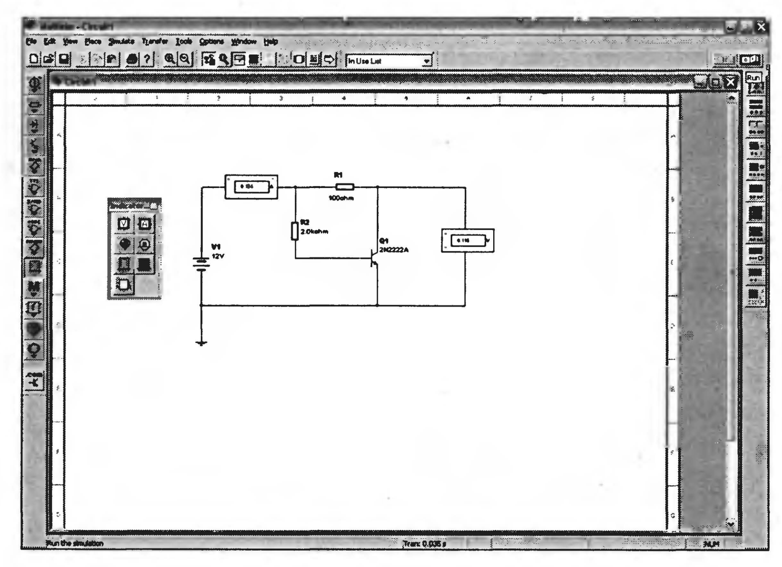


Рис. 3.8. Измерение напряжений

Усилительный каскад на транзисторе

Если открыть любой учебник по схемотехнике усилителей, можно там же найти и методики расчета усилителей. В учебнике обязательно будет приведена классификация усилительных каскадов по способу включения транзистора как активного элемента и свойства каскадов при разных способах включения транзисторов. Однако чаще всего применяют включение транзистора с общим эмиттером. Это означает, что эмиттер служит общим выводом для входной и выходной цепи. Тем, кто интересуется теоретическими аспектами вопроса, кому хотелось бы методично во всем разобраться, я могу порекомендовать несколько книг.

1. Герасимов, Мигулин, Яковлев. Расчет полупроводниковых усилителей и генераторов. Киев, 1961.
2. Воронков, Овечкин. Основы проектирования усилительных и импульсных схем на транзисторах. Москва, 1973.

3. П.Шкритек. Справочное руководство по звуковой схемотехнике. Москва, 1991.

Мы же воспользуемся преимуществами, которые получаем от использования компьютера.

Несколько слов о том, почему мне хочется начать с усилительного каскада на транзисторе (рис. 3.9). Мне кажется, что знание работы транзистора закладывает основу понимания и аналоговой техники, и цифровой, включая микропроцессорную, техники. Конечно, сегодня при разработке электронных устройств или ремонте использование микросхем подразумевает знание не того, как устроена микросхема, а того, какими свойствами она обладает (ее параметров), для чего предназначена и какие сигналы использует. В этом смысле сегодняшняя работа с электроникой ближе к программированию на объектно-ориентированных языках, в отличие от программирования с использованием процедурных языков. Некоторые преподаватели информатики даже считают, что знания процедурных языков мешает быстрому освоению современного программирования. Возможно, так. Но быстрое освоение в узкой области знаний рано или поздно может завести в тупик. Я так думаю, но спорить не готов.

Вот так будет выглядеть усилительный каскад с общим эмиттером. В качестве источника сигнала использован генератор синусоидального напряжения, а сигналы на входе и выходе усилителя мы наблюдаем с помощью осциллографа. Немного изменим нашу схему, как показано на рис. 3.10.

В схеме появились два вольтметра переменного тока для измерения величины сигнала. Как видно из рисунка (по показаниям вольтметров), сигнал на входе усилителя – около 6 мВ, а на выходе – 2 В. Отношение выходного сигнала к входному – это коэффициент усиления каскада по напряжению. Разделив 2 В на 6 мВ, мы получим, что Кн = 333. Усиление по напряжению интересует нас довольно часто.

Здесь уместно пояснить один момент, касающийся единиц измерения. Усиление мы измеряем в относительных единицах как отношение выходного напряжения к входному

Рис. 3.9. Усилительный каскад на транзисторе с общим эмиттером

Рис. 3.10. Измерение входного и выходного напряжения сигналов

(Ку = Uвых / Uвх). Есть и другие единицы измерения – децибелы. Формула перехода выглядит так – Ку, дБ = 20lg Ку или Ку, дБ = 20lg (Uвых / Uвх).

Зачем нужны другие единицы? Общее усиление двухкаскадного усилителя равно произведению коэффициентов усиления. А мы знаем из свойств логарифмов, что логарифм произведения равен сумме логарифмов. Таким образом, вместо сложной операции умножения можно применить более простую операцию – сложение. Порой это удобно.

Вернемся к схеме. Для чего служат ее элементы? Сопротивление R1 определяет базовый ток транзистора, ток коллектора которого равен некоторому параметру транзистора, называемому статическим коэффициентом усиления транзистора, умноженному на его базовый ток. Расчет каскада «на вскидку» можно провести так – мы определяем, что в отсутствие сигнала напряжение на коллекторе должно быть равно половине напряжения питания (чтобы симметричный сигнал мог усиливаться максимально, но без искажений). Зная величину сопротивления R2 = 5 кОм (сопротивления нагрузки транзистора) и величину напряжения (12 В / 2 = 6 В), можно легко определить необходимый ток коллектора. Ток коллектора равен половине напряжения питания, деленной на сопротивление нагрузки (6 В / 5 кОм = 1,2 мА). Ток коллектора связан с током базы величиной статического коэффициента усиления транзистора по току в схеме с общим эмиттером (Вст = 200 для данного транзистора 2N2222): Iк = Вст × Iб, где Iк – ток коллектора, Iб – ток базы.

Таким образом, ток базы должен быть в 200 раз меньше, то есть 6 мкА. Через резистор R1 должен протекать ток в 6 мкА, а напряжение на нем должно быть равно напряжению питания минус напряжение база–эмиттер транзистора. Для расчетов «на вскидку» последним можно пренебречь, приняв напряжение на резисторе равным напряжению питания. Тогда величина резистора R1 будет равна отношению напряжения питания к току базы транзистора – 12В / 6мкА (то есть, 2 МОм). Что мы и сделали. Остался вопрос, почему величина резистора R2 выбрана равной 5 кОм? Эта величина обычно определяется входным сопротивлением следующего каскада. Например, можно считать, что наш усилитель будет подключен

к усилителю мощности. Линейный вход усилителя мощности имеет, примерно, такие параметры: входное напряжение – 250 мВ, входное сопротивление – 47 кОм. Если выходное сопротивление каскада будет на порядок ниже, то есть 4,7 кОм, входное сопротивление усилителя мощности не будет мешать работе каскада. Выходное же сопротивление каскада не превысит сопротивления нагрузки транзистора, равного 5 кОм.

Усиления каскада в 300 единиц по напряжению, казалось бы, достаточно для многих целей. Так сигнал обычного динамического микрофона, который будет включен вместо генератора, имеет порядок 1 мВ, а выходной сигнал в 300 мВ соответствует уровню стандартного линейного входа усилителя мощности. Чего нам еще желать?

Но если мы посмотрим на реальные схемы, то увидим, что они имеют несколько иной вид (рис. 3.11).

Что получилось с усилением? Кн = 100мВ / 5мВ (Кн = 20). Усиление снизилось более чем в 10 раз. Зачем же нам нужны лишние детали – резисторы R3 и R4, и потеря усиления?

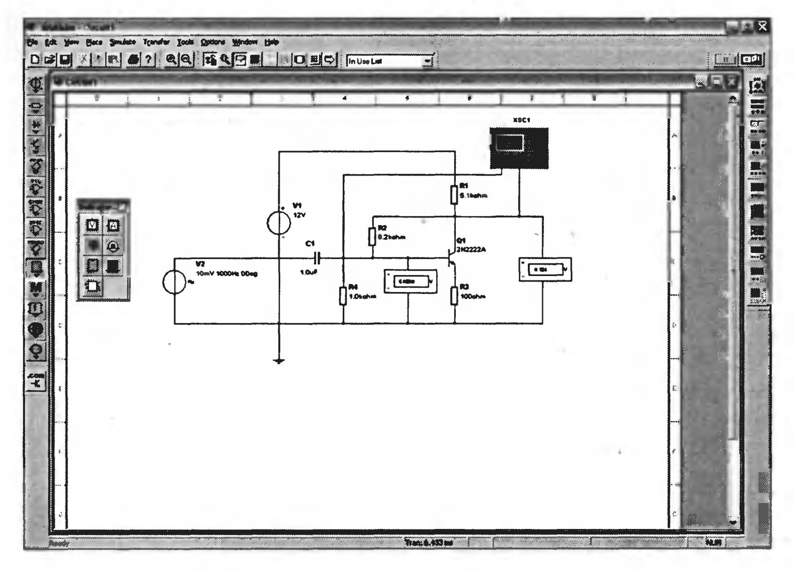


Рис. 3.11. Типовая схема каскада с общим эмиттером

Дело в том, что, в первую очередь, полупроводники очень чувствительны к изменению температуры. Это их свойство часто используют, изготавливая из полупроводников датчики температуры. У транзистора с изменением температуры окружающей среды изменится ток коллектора, что сместит напряжение на коллекторе и может привести к искажению сигнала. Аналогично действует изменение напряжения питания. Попробуйте повлиять на ток коллектора, меняя величину сопротивления резистора R1, и понаблюдайте за формой сигнала на экране осциллографа. Этого же можно добиться, меняя температуру в диалоговом окне условий моделирования (**Simulate ➡ Analyses ➡ Temperature Sweep**). Для этого нужно проставить новое значение температуры в строке **Values** открывающегося диалогового окна и щелкнуть кнопку **Accept**.

Дополнительные элементы схемы призваны стабилизировать параметры усилительного каскада. Если обратиться к рис. 3.9, можно заметить, что входной и выходной сигналы находятся в противофазе. В нашей схеме (рис. 3.11) резистор R3 в сочетании с входным сопротивлением каскада образует параллельную обратную связь, а резистор R2 – резистор последовательной обратной связи. Обе обратные связи – отрицательные по постоянному току – стабилизируют все параметры усилительного каскада.

Посмотрим, что произойдет, если ток коллектора возрастет. Напряжение на коллекторе транзистора падает, ток через резистор R3 уменьшается, что приводит к уменьшению базового тока транзистора, а, следовательно, и тока коллектора, равного величине базового тока, умноженного на статический коэффициент усиления транзистора.

Похожие изменения происходят и при наличии резистора R2. Увеличение тока коллектора приводит к увеличению падения напряжения на этом резисторе. Ток базы зависит от напряжения между базой и эмиттером транзистора, которое при увеличении падения напряжения на резисторе R2 уменьшается (оно равно разности напряжения между базой и общим проводом и падением напряжения на резисторе R2). С уменьшением напряжения между базой и эмиттером уменьшается ток базы, а, следовательно, ток коллектора.

Есть еще один из достаточно важных параметров каскада, на который отрицательная обратная связь влияет благотворно.

Верхняя граничная частота усилителя на рис. 3.9 составляет примерно 2,3 МГц (частота, на которой коэффициент усиления уменьшается на 3 децибела). Верхняя граничная частота усилителя на рис. 3.11 – более 16 МГц.

Для определения этой частоты можно построить частотную характеристику усилителя – зависимость усиления от частоты. В средней части частотной характеристики график идет горизонтально, а после верхней граничной частоты происходит спад усиления в 20 дБ на декаду (то есть, изменение частоты в 10 раз приводит к спаду усиления на 20 дБ или в 10 раз). Частотную характеристику можно строить по точкам. Изменив частоту, определить выходное напряжение, вновь изменить частоту и измерить выходное напряжение и т.д. Программа Multisim предлагает воспользоваться для этой цели прибором – плоттером Боде. При работе с прибором не следует забывать, что он работает тогда, когда на входе схемы включен генератор переменного напряжения (его параметры не существенны). Кроме того, включив схему, можно не обнаружить частотной характеристики на экране плоттера Боде. Скорее всего, причина в том, что она выше выделенного окна. Изменяя параметр F вертикального отклонения, можно найти пропажу (рис. 3.12).

Кому-то подобные простые объяснения могут показаться излишне простыми. Но почему-то в сложных ситуациях именно об этих простых вещах вспоминаешь в последнюю очередь. Впрочем, это дело вкуса.

Так выглядит частотная характеристика усилительного каскада. Для получения фазовой характеристики на плоттере Боде следует включить клавишу **Phase** в верхней части прибора.

Знание фазовой характеристики усилительного каскада позволяет определить устойчивость усилителя после введения отрицательной обратной связи. На граничной частоте фаза сигнала изменяется на 45° и продолжает изменяться со скоростью 45° на декаду (относительно фазы на входе). При

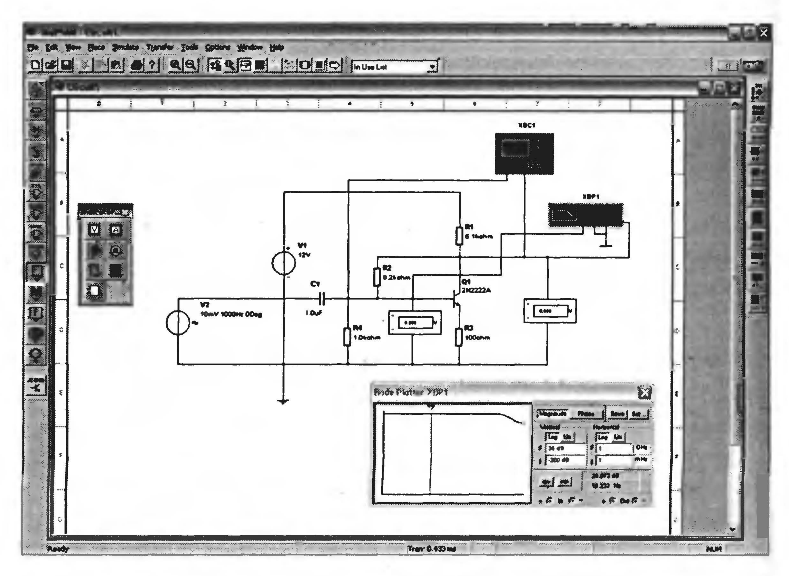


Рис. 3.12. Частотная характеристика каскада на плоттере Боде

этом отрицательная обратная связь на некоторой частоте превращается в положительную. Если при этом усиление больше единицы, усилитель превращается в генератор. Поскольку часть сигнала с выхода усилителя приходит на вход в фазе с входным сигналом, складывается с ним, увеличивая выходной сигнал, часть которого складывается с входным...

Пожалуй, этого краткого введения будет достаточно. Могу только добавить, что первые иллюстрации сделаны в новой версии программы, которая, что бросилось мне в глаза в первую очередь, весьма пополнилась в части используемой измерительной аппаратуры. Осциллограф не только выглядит, как настоящий, но и позволяет покрутить ручки, включить меню настроек на экране. Используемые микросхемы пополнились, например, PIC-контроллерами. Программа, как и предыдущие версии, очень наглядна и удобна для решения быстро возникающих неотложных проблем. Имеет интегрированный компоновщик PCB (рис. 3.13).

Единственный существенный для меня недостаток программы – стоимость полной профессиональной версии больше 4000 долларов. А демо-версия (именно ее я использовал

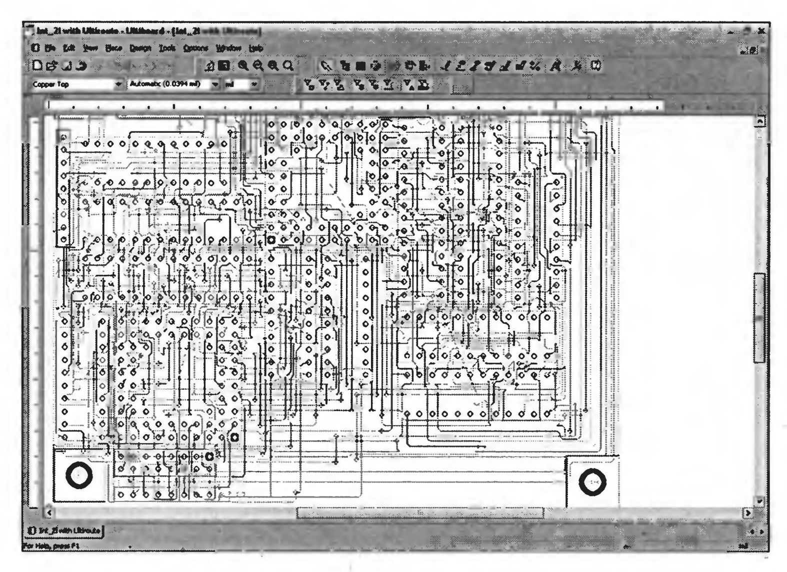


Рис. 3.13. Вид платы в программе Ultiboard 9

для иллюстрации) завершает работу на моем компьютере через несколько дней, и в отличие от предыдущих версий, работающих без сохранения файла, не имеет приборного дополнения. Увы!

CircuitMaker 2000

Эта программа не менее удобна, чем предыдущая. Она несколько дешевле. Всего 1500 долларов, что не мешает мне забыть о возможности ее приобретения. Программа помогает строить как цифровые схемы, так и работать с аналоговыми (рис. 3.14).

И нельзя сказать, что просмотр сигналов в этой программе неудобен. Наоборот, все удобно и наглядно (рис. 3.15).

Программа поможет вам проверить идеи при создании охранной подсистемы. Не всегда очевидно, каким образом организовать работу этой подсистемы. «На вскидку» решение выглядит просто: сработал датчик – бей тревогу. Без продуманного подхода реализация может дать обратный эффект, в первую очередь, за счет ложных срабатываний.

Рис. 3.14. Клавиатура в программе CircuitMaker

Рис. 3.15. Усилитель в программе CircuitMaker

Затем последуют срабатывания системы, если вы забудете (или не успеете) снять ее с режима охраны и т.п.

Самым лучшим решением в подобной ситуации станет тщательное продумывание, в котором поможет программа CircuitMaker (рис. 3.16).

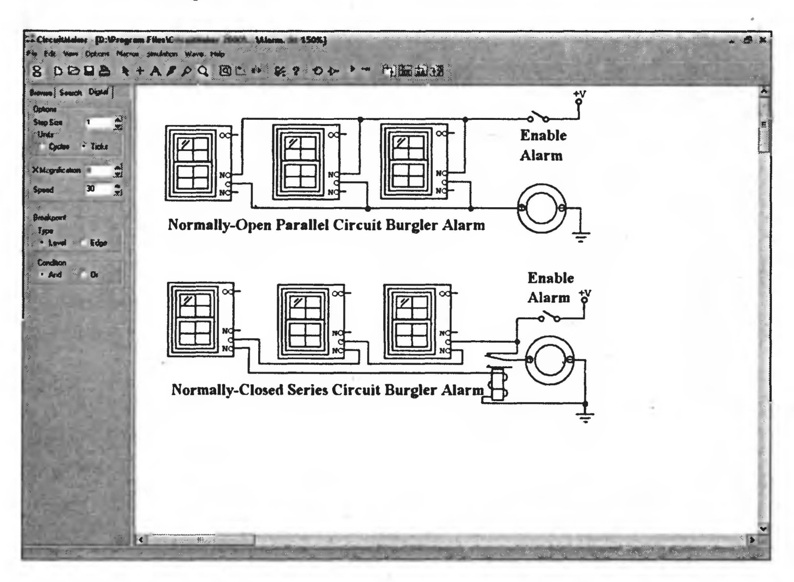


Рис. 3.16. Подсистема охраны в программе CircuitMaker

Как и предыдущая, она имеет часть, предназначенную для создания печатной платы. Современная любительская технология позволяет сделать печатную плату очень хорошего качества. Если использовать поверхностный монтаж, выполнение платы может мало отличаться от промышленного. Я имею в виду использование лазерного принтера для нанесения рисунка. В этом случае наличие возможности выполнить разводку в той же программе, где разработана схема, скорее обязательно, чем желательно (рис. 3.17).

Я не сомневаюсь, что для многих из вас высокая стоимость программ не является препятствием для их использования. И не думаю, что вышеупомянутыми программами ограничивается выбор. Я только хотел подчеркнуть, что, используя

микроконтроллеры, не следует забывать о микросхемах попроще: транзисторах и резисторах, конденсаторах и трансформаторах.

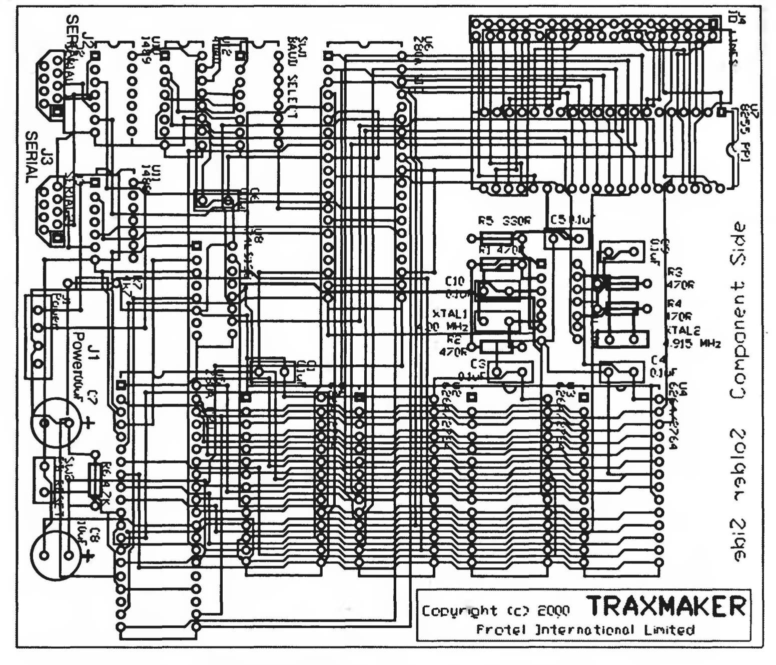


Рис. 3.17. Вид печатной платы в программе TraxMaker

Даже такой возможностью, как просто отрисовать схему, не следует пренебрегать. Одно дело мысленно нарисовать схему, в которой легко ошибиться, скажем, забыв, что счетчик – десятичный, а не двоичный. Другое дело – нарисовать ее на бумаге. Правда, у меня бумага после недолгой работы со схемой превращается в грязное подобие решета – следствие многократных переделок. Для рисования есть удобные графические средства. Когда-то я использовал AutoCAD LT, но возможность нарисовать и проверить работу, хотя бы в первом приближении, мне кажется очень полезной.

Поскольку использовать дорогостоящие программы я не могу, то пытаюсь отыскать программы, которые доступны

мне по цене. В частности, в последнее время все активнее использую операционную систему Linux, для которой программное обеспечение стоит много дешевле, и существует великое множество бесплатных программ. Следующая программа доступна в двух версиях: для Windows и для Linux. Она, как мне кажется, больше предназначена для разработки микросхем, судя по началу руководства к ней. Кстати, руководства весьма объемистого и подробного.

Electric

Сразу оговорюсь: хотя я подозреваю, что в программе можно моделировать великое множество схем, я попробовал работу только простейшей схемы. В настоящий момент программа мне не требуется, и я не хочу тратить время на ее освоение до того момента, когда в этом будет необходимость. Программу можно найти на сайте http://www.staticfreesoft.com.

Для работы программы необходимо установить пакет Java (в последних версиях это не обязательно). Полагаю, как и у меня, он найдется в дистрибутиве под именем jre-1.5.0-1asp.i386.rpm или аналогичным. Программы для установки в Linux имеют расширение .rpm, но не для всех дистрибутивов. После установки пакета можно запускать программу. Я установил ее первоначально в своей домашней папке, но затем перенес в папку /usr/etc. Для запуска создал кнопку запуска (ярлык) с командой java-jar /usr/etc/electric.jar-mdi, где последние символы (-mdi) означают, что я хотел бы видеть все части программы в едином окне, что необязательно.

После запуска программы появляется окно, в котором можно сразу обратиться к руководству пользователя, которое находится в разделе **Help ➡ User's Manual** (рис. 3.18).

Надеюсь, вам это руководство поможет больше, чем мне. Оно достаточно подробно, хорошо иллюстрировано, удобно в обращении. Но написано, как мне кажется, для пользователей Windows, хотя есть все уточнения и для Мас-, и для nix-пользователей.

Чтобы убедиться в том, что программа работает, можно сделать следующие, возможно, и неправильные шаги.

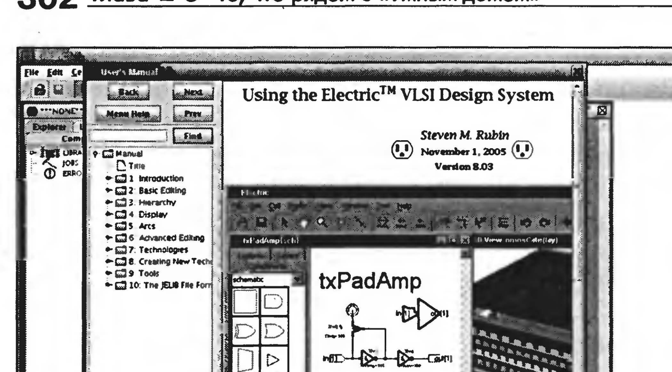


Рис. 3.18. Окно руководства пользователя в Electric

Сохраним библиотеку с помощью меню **File ➡ Save Library As...**, где в открывшемся диалоговом окне выберем заранее созданную в своей домашней папке подпапку с названием Electric_projects. Открывать папку лучше кнопкой **Open**. Затем изменим noname.jelib на любое имя первого проекта, например first.jelib, и щелкнем появившуюся кнопку **Save**. Выберем в меню раздел **Cell ➡ New Cell...** В диалоговом окне, которое появится, выберем **schematic** и впишем имя **first** в окне для имени первой схемы. Подтвердим выбор кнопкой **OK**.

Надпись об отсутствии **Cell** пропадает, вместо нее появляется курсор в виде крестика. Выберем слева в проводнике ярлычок **Components**, а в открывшемся меню компонентов – схему **И** (на рис. 3.19 видно меню компонентов). Схема И – верхняя правая, ее мышкой перенесем на рабочее поле чертежа:

Теперь, если поводить маркером по картинке, видно, как на полуокружности задней части объекта появляются три крестика: один – в верхней части, второй – посередине, третий – в нижней части. Если выделить верхнюю метку левой

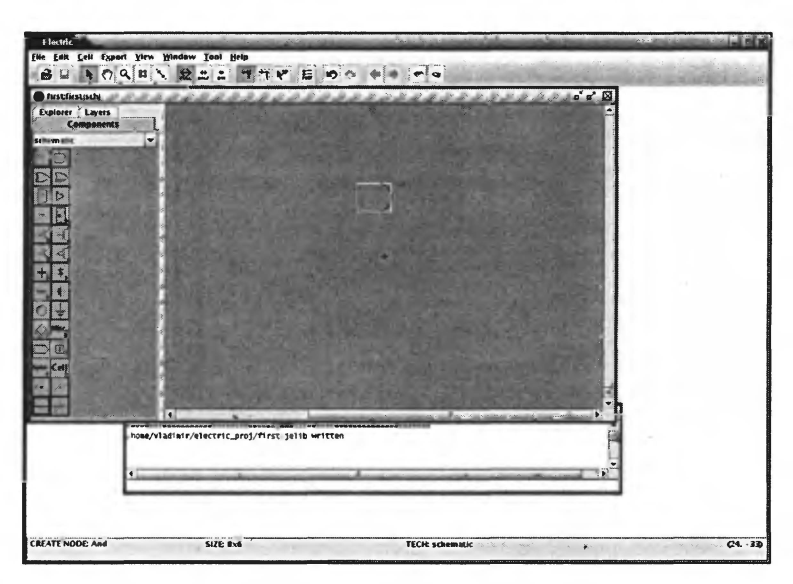


Рис. 3.19. Окно с панелью компонентов

кнопкой мыши, а затем, удерживая правую кнопку, провести линию в сторону базовой линии фигуры, которую, возможно, потом нужно будет выделить, то (иногда мне удается добиться этого, просто, сразу щелкнув правой кнопкой мышки в момент, когда маркер выделяет метку) получим результат, показанный на рис. 3.20.

В разделе меню **Export** ➡ **Create Export**, в диалоговом окне для Export Name зададим «a». В окошке же пониже выберем **input**.

И по доброй уже традиции щелкнем **OK**. Теперь в разделе **Tool** ➡ **Simulation (Built-in)** ➡ **ALS: Simulate Current Cell**, щелкнув все в правильной последовательности, получим результат, показанный на рис. 3.21.

Теперь можно закрыть окно наблюдения; выделить нижнюю метку аналогично верхней и через разделы меню **Export** и **Tool** получить отображение обоих входов. Выделив метки входов курсором и щелчком левой кнопкой мыши, правой кнопкой можно «удлинить» входы микросхемы. Можно перенести буквы, обозначающие входы, вперед, чтобы микросхема приобрела более привычный вид. Для упрощения

Рис. 3.20. Окно с выбранной схемой **И** и диалогом Export

Рис. 3.21. Окно наблюдения в программе Electric

последней процедуры я на вкладке проводника проекта Explorer в верхней части левого окна раскрываю проект first (Current), выделяю название окна схемы firs (sch) правой кнопкой мыши. Появляется раскрывающееся меню, в котором я выбираю раздел **Edit in New Window**. После открытия окна можно закрыть предыдущее, а в новом окне легче выделить буквы и перетащить их в новое место.

Для оживления окна наблюдения выделяем вход, например «b» (я полагаю, что раздел меню **Tool ➡ Simulation (Built-in) ➡ ALS: Simulate Current Cell** уже пройден), через выделение метки входа «b» на дуге. Затем в меню выбираем **Tool ➡ Simulation (Built-in)**, где, в свою очередь, щелкаем пункт **Set Clock on Selected Signal**. Соглашаемся с предложенной частотой (**OK**). Вид окна наблюдения сразу изменится (рис. 3.22).

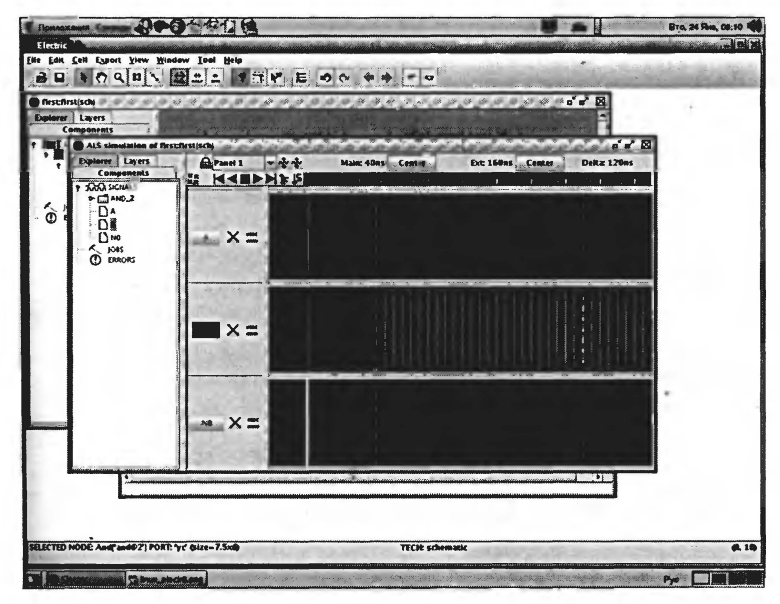


Рис. 3.22. Окно наблюдения при включенном тактовом генераторе

Воспользуемся клавишами управления, подобными клавишам аудио и видеопроигрывателей под надписью **Panell**, запустим время клавишей **Play**. И, когда подвижной маркер дойдет до временной отметки 60–80 ns, остановим его клавишей стоп. Мне приходится быстро нажимать ее несколько раз. Теперь вернемся к чертежу, выделим вход «а», войдем в меню **Tool ➡ Simulation (Built-in)** и выберем пункт **Set Signal High at Main Time**. Вид окна наблюдения изменится, что, с моей точки зрения, свидетельствует о правильном отображении событий. Сигнал на выходе появляется в тот момент, когда он принимает значение логической «1» на входе «а».

Повторно запуская маркер клавишей «**Play**» и останавливая его через некоторое время у отметки, например, близкой к 120 ns (мы оставили вход «а» на схеме в выделенном состоянии), входим в раздел меню **Tool ➡ Simulation (Built-in)**, но теперь выбираем пункт **Set Signal Low at Main Time**. Вид окна наблюдения меняется, и мы получаем вариант работы схемы, управляемой по входу «а» и тактовым генератором по входу «b». Если временной интервал задавать с помощью таймера на микросхеме 555, думаю, он будет работать с нужными временами, а этих таймеров можно сделать несколько. Если, в свой черед, к выходу микросхемы И подключить светодиод, ИК или АЛ307, как это было в первой части книги, а частоту тактового генератора выбрать равной 37 кГц, можно придумать альтернативную схему излучения ИК-команды.

По крайней мере, мы получили пачку импульсов, готовую к употреблению (рис. 3.23).

Добавлю еще одно замечание – программа позволяет создать, подобно pcb-программе, вид печатной платы, но я не пробовал.

Сопряжение управления

Наличие нескольких телевизоров в сегодняшнем быту далеко не редкость. Телевизор в гостиной, DVD-проигрыватель, видеомагнитофон, телевизор на кухне, в спальне. Как лучше согласовать подключение всех источников видеосигнала, можно

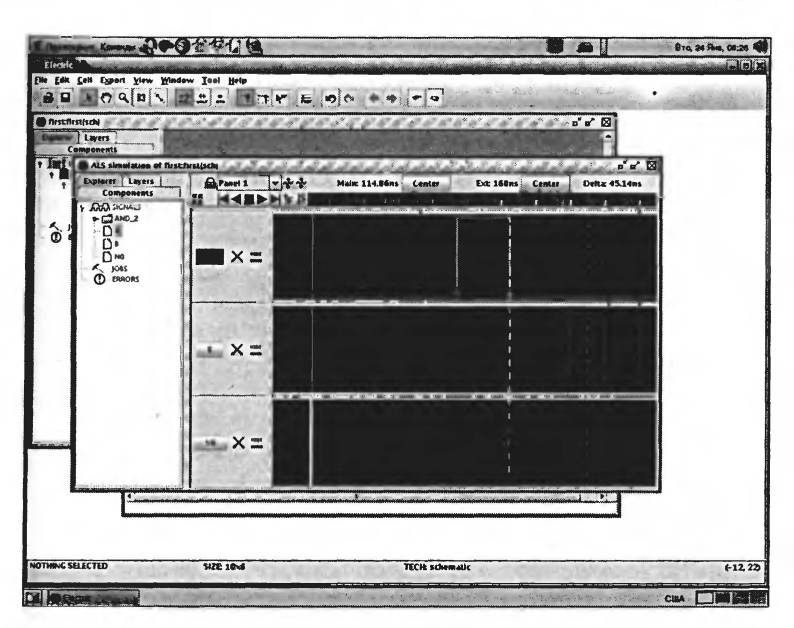


Рис. 3.23. Окончательный результат наблюдения

посмотреть в Приложении. В системе «Умный дом» распределение видеосигнала по всем помещениям, естественно, сопровождается распределением управления (рис. 3.24). Как это может выглядеть?

Из приведенной схемы ясно, что DVD-проигрыватель, расположенный в гостиной, управляется дополнительно из спальни и кухни.

Не вдаваясь в детали устройства видеокоммутатора, я хочу предложить вам рассмотреть следующий сценарий.

Простуда уложила вас в постель. Пользуясь случаем, вы решили посмотреть новый фильм, который купили накануне. Уютно расположившись в спальне, вы смотрите фильм, а домашние, чтобы не беспокоить вас, досматривают очередную серию очередного сериала на кухне.

Серия закончилась. Ваши домашние вспомнили о фильме, который собирались посмотреть вчера и даже установили в проигрыватель, но не успели посмотреть. Они нажимают на пульте управления кнопочку DVD и... Как и положено,

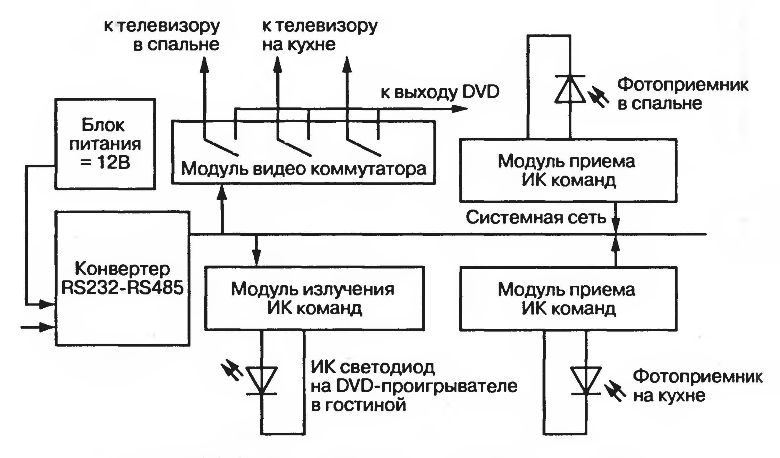


Рис. 3.24. Подключение DVD-проигрывателя к телевизорам

команда power, воспроизводимая модулем излучения ИК-команд, выключит DVD-проигрыватель на самом интересном месте, когда интрига фильма заставляет вас забыть о простуде. Система «Умный дом» в данном случае позаботится о том, чтобы вам о ней напомнить.

Но я знаю, что вы человек предусмотрительный, и, хотя нет единого мнения, как лучше организовать работу устройств, управляемых из разных помещений, вы предусмотрели в программе управления подобную ситуацию, нарисовав, например, алгоритм управления, представленный на рис. 3.25.

Но к телевизору вы можете подключить и видеомагнитофон, а также использовать телевизор в качестве акустической зоны, подключая его к музыкальному центру. В этом случае алгоритм усложнится. Этому будет способствовать и наличие большого количества помещений с развитой системой средств управления и устройств, управляемых из разных помещений.

После завершения программы управления системой она приводится в действие. При этом трудно разобраться во всех сложных ситуациях. Конечно, вы постарались предусмотреть



Рис. 3.25. Алгоритм управления DVD–проигрывателем

все, когда разрабатывали алгоритм управления. Но все ли вы предусмотрели?

В аналогичной ситуации я поступаю следующим образом. Нарисовав алгоритм, я проверяю его с помощью вспомогательной программы, для создания которой можно использовать любую современную среду программирования. Я использовал Delphi и KDevelop с одинаковым успехом. Можно воспользоваться любой средой программирования и любым языком. Единственное, как мне кажется, важное условие – легкое создание графических примитивов и управление их свойствами. В данном случае программа не будет конечной целью, она – лишь средство избавить вас от сомнений в правильности принятого решения.

Программа, в сущности, описывает ваш алгоритм, а графическая часть изображает устройства управления, управляемые устройства, возможно, флаги и переменные. Выглядеть работающая программа может, как показано на рис. 3.26.

На рис. 3.26 изображены устройства управления и управляемые устройства, которые при включении меняют цвет, флаги и т.д.

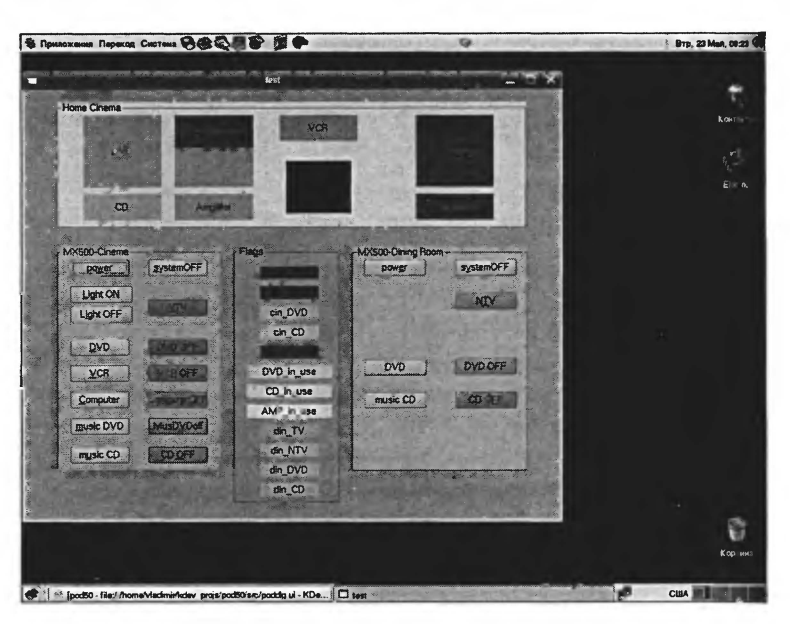


Рис. 3.26. Проверка алгоритма управления

Ниже я привожу фрагмент программы, чтобы показать, что в ней используются только графические примитивы и изменение их свойств. Написать такую программу не составит труда.

```
void test::cinTV_ONSlot()
{
cin_TV = true;
cin_NTV = true;
cin_AVproc = true;
cin_AVproc_f->setBackgroundColor("Red"); // AVprocessor Cinema
cin_ntv_f->setBackgroundColor("Red"); // NTV Cinema
cin_tv_f->setBackgroundColor("red");
extron_eqip1->setText("EXTRON\n NTV->Cinema");
extron_eqip1->setBackgroundColor("red"); // EXTRON NTV Cinema
proj_equip->setText("Projector\n NTV-Composit");
proj_equip->setBackgroundColor("red");
av_proc_equip->setBackgroundColor("red");
av_proc_equip->setText("AV Processor\n NTV-3D");
```

```
Light->setBackgroundColor("Black");
}

void test::cinTV_OffSlot()
{if(cin_AVproc){
cin_AVproc = false;
cin_AVproc_f->setBackgroundColor("LightGrey"); // AVProcessor
cin_TV =false;
cin_tv_f->setText("cin_TV");
cin_tv_f->setBackgroundColor("LightGrey"); // TV
cin_NTV = false;
cin_ntv_f->setText("cin_NTV");
cin_ntv_f->setBackgroundColor("LightGrey"); // NTV
extron_eqip1->setText("EXTRON");
extron_eqip1->setBackgroundColor("LightGrey");
extron_eqip1->setText("EXTRON\n NTV->Cinema"); //Extron Cinema
proj_equip->setText("Projector\n NTV-Composit");
proj_equip->setBackgroundColor("LightGrey");
av_proc_equip->setText("AV Processor\n NTV-3D");
av_proc_equip->setBackgroundColor("LightGrey");
Light->setBackgroundColor("White");

if (cin_DVD){
 cin_DVD = false;
 cin_dvd_f->setBackgroundColor("LightGrey"); // DVD
 DVD_in_use = false;
 dvd_inuse_f->setBackgroundColor("Yellow"); // DVD
 dvd_equip->setText("DVD");
 dvd_equip->setBackgroundColor("LightGrey");
 }
 if (cin_VCR){
  cin_VCR = false;
  vcr_equip->setBackgroundColor("LightGrey");
  vcr_equip->setText("VCR"); // VCR
 }
 if (cin_CD){
 cin_CD = false;
 cin_cd_f->setBackgroundColor("LightGrey"); // CD
 CD_in_use = false;
 cd_inuse_f->setBackgroundColor("Yellow"); // CD
 cd_equip->setText("CD");
```

```
 cd_equip->setBackgroundColor("LightGrey");
  }
  }
}

void test::cinNTV_onSlot()
{
if (cin_TV){
  if (!cin_DVD){
   cin_NTV = true;
   cin_ntv_f->setBackgroundColor("Red"); // NTV Cinema
   extron_eqip1->setText("EXTRON\n NTV->Cinema");
   extron_eqip1->setBackgroundColor("red"); // EXTRON NTV
Cinema
   proj_equip->setText("Projector\n NTV-Composit");
   proj_equip->setBackgroundColor("red");
   av_proc_equip->setText("AV Processor\n NTV-3D"); // 3D-
sound
   }
  else {
   cin_NTV = true;
   cin_ntv_f->setBackgroundColor("Red"); // NTV Cinema
   cin_DVD = false;
   cin_dvd_f->setBackgroundColor("LightGrey");  // DVD
   DVD_in_use = false;
   dvd_inuse_f->setBackgroundColor("Yellow"); // DVD
   extron_eqip1->setText("EXTRON\n NTV->Cinema"); // Extron
Cinema
   proj_equip->setText("Projector\n NTV-Composit");
   av_proc_equip->setText("AV Processor\n NTV-3D"); // 3D-
sound
   dvd_equip->setText("DVD");
   dvd_equip->setBackgroundColor("LightGrey"); // DVD
   }
}
}

void test::cinDVD_onSlot()
{
if (cin_TV){
   if (!DVD_in_use){
   cin_NTV = false;
```

```
  cin_ntv_f->setBackgroundColor("Light Grey"); // DVD_in_use
= true;
  dvd_inuse_f->setBackgroundColor("red"); // DVD
  cin_DVD = true;
  cin_dvd_f->setBackgroundColor("red"); // DVD
  dvd_equip->setBackgroundColor("red");
  dvd_equip->setText("DVD for Cinema"); // DVD
  av_proc_equip->setText("AV Processor\n DVD-Dolby");
  av_proc_equip->setBackgroundColor("Red"); // DVD
DolbyDigital
  proj_equip->setText("Projector\n DVD-S_video"); // DVD
   }
   else{
  cin_NTV = false;
  cin_ntv_f->setBackgroundColor("Light Grey"); //
av_proc_equip->setText("DVD");
  av_proc_equip->setText("AV Processor\n DVD-Dolby"); // DVD
DolbyDigital
  proj_equip->setText("Projector\n DVD-S_video"); // DVD
   }
 }
}

void test::cinDVD_offSlot()
{if ((cin_TV)&&(!cin_VCR)&&(!cin_COMP)){
  if (cin_DVD){
  cin_NTV = true;
  cin_ntv_f->setBackgroundColor("Red"); // NTV Cinema
  cin_DVD = false;
  cin_dvd_f->setBackgroundColor("LightGrey"); // DVD
  DVD_in_use = false;
  dvd_inuse_f->setBackgroundColor("Yellow"); // DVD
  proj_equip->setText("Projector\n NTV-Composit");
  av_proc_equip->setText("AV Processor\n NTV-3D"); // 3D-
sound
  dvd_equip->setText("DVD");
  dvd_equip->setBackgroundColor("LightGrey"); // DVD
  }
```

При нажатии клавиши пультов управления я «включаю и выключаю» устройства (меняю цвет), устанавливаю и сбрасываю флаги – словом, контролирую свое нехитрое хозяйство.

Запустив программу, я начинаю беспорядочно щелкать по всем клавишам устройств управления. Моя задача – проверить, не возникает ли тупиковая ситуация, подобная описанной выше, при каких-либо управляющих сигналах. Используя этот нехитрый прием, я убедился, что допускал ошибки в алгоритмах даже после их тщательной проверки на бумаге. В первую очередь, это ошибки, которые я называю для себя ошибками последовательности действий. Описывая алгоритм, зачастую укрупняешь операции. Например, в алгоритме, приведенном выше, я записал включить DVD, установить флаг. В действительности команда включить DVD распадается на ряд команд, каждая из которых может ветвиться: вывести устройство из режима ожидания, открыть лоток для установки диска, закрыть лоток, включить меню, выбрать из меню режим просмотра и т.д. Каждому действию соответствует своя последовательность нажатия клавиш на пульте. Когда «крутишь» алгоритм на бумаге, легко проглядеть возможные ветвления и ошибки в последовательности. Особенно, если работа, как это часто бывает, происходит в условиях дефицита времени. Использование программы для проверки программы (возможно, есть математические методы, решающие эти проблемы, но нет времени с ними разобраться) позволяет до введения в систему неких действий проверить, не приведут ли они к нежелательным результатам. И удалить, как минимум, тупиковые ситуации.

Сами по себе эксперименты в этой области достаточно интересны. Но, если вы от экспериментов перешли к действиям, не забывайте, что выявление дефектов в действующей системе сложнее, чем на стадии проекта. Возвращаться к уже сделанному, когда первоначальный интерес сменился желанием пользоваться системой без проблем, совсем не хочется. А раздражение, которое может вызывать «поехавшая крыша Умного дома», не стоит нескольких дополнительных усилий, приложенных в начале работы.

Кроме того, вы все делаете для себя. Я уже говорил, что не все придерживаются точки зрение, что сопряжение управления следует делать. Вам тоже, возможно, это покажется

лишним. Согласен. Но прием, о котором я рассказал, может быть полезен и в других случаях.

Смешанные системы

Вы завершили эксперименты с модулями, предложенными в начале книги, и отважились на создание собственной системы на базе этих модулей. Вы все тщательно продумали, опробовали, реализовали. Осталось насладиться результатами. Но вот беда – вам приглянулся угловой диван для гостиной, понравилось, что у него есть бар и полка, на которую можно положить роман для чтения на сон грядущий. В полке есть лампочка, которую вы включаете, когда читаете роман. Розетка для включения лампочки есть, выключатель разместился на проводе, включаемом в розетку. Но вам не нравится, что этот выключатель вы не можете добавить в систему. Это раздражает всякий раз, когда вы им пользуетесь.

Вместе с тем, вам не нравится идея прокладывать новые провода к дивану. Вы только завершили прокладку проводов. Как поступить в этом случае?

Есть разные пути решения – использовать радиоканал, ИК-канал для передачи команд управления. Если же вдобавок вы хотите использовать возможности управления яркостью, но не хотите самостоятельно изготавливать диммер, то самое лучшее решение – применить устройство другой системы. Об этом я хочу рассказать немного подробнее.

Если вам приходится использовать устройства другой системы, в первую очередь, вам помогут специалисты, представляющие продукцию той фирмы, на которой вы остановили свой выбор. Например, вы используете систему X10, но в качестве основной панели управления хотите использовать современную красивую сенсорную панель фирмы Crestron. Уверен, специалисты фирмы, у которой вы приобрели оборудование X10, возможно, проектировали вашу систему, и помогут вам в этом. Так фирма «Умный дом» предлагает большое количество устройств X10, но одновременно она представляет и продукцию Crestron.

Но вернемся к «суете вокруг дивана». Вы не хотите менять всю систему, но желаете использовать модуль X10. В этом случае я советую использовать компьютерный интерфейс для управления этим модулем. Интерфейс будет включен, если вы используете управляющую программу на компьютере или любом удобном месте, воспользовавшись прямой связью модулей в системе.

Для решения проблемы управления модулем X10 вам потребуется дополнительно подключить конвертер RS485–RS232 к системной сети, а к нему – интерфейс компьютера для системы X10. Использовать при этом придется систему команд протокола X10.

В некоторых случаях дополнительные устройства могут работать по интерфейсу RS485. Если известна система команд управления этим устройством, то трудности при интеграции устройства в систему, сделанную своими руками, не должны возникнуть. Если система команд не известна, но команды управления достаточно просты, можно попытаться «подсмотреть» команды управления. Используя любую программу, работающую с COM-портом, готовую или написанную самостоятельно на основании тех рекомендаций, что есть в книге, подключите компьютер через конвертер RS232–RS485 к линии и подайте команды от штатного устройства управления. Программа для COM-порта должна работать на чтение сигналов линии.

Такой нехитрый прием позволяет узнать команды управления, которые можно использовать в дальнейшем. Иногда отсутствие команд управления не связано с тем, что они скрываются изготовителем оборудования. Скорее, оборудование, которое вы хотите использовать в своей системе, предназначается для использования в другой системе. Или протокол работы слишком сложен для быстрого повторения вне системы. В последнем случае, если вы располагаете достаточными финансовыми возможностями, ваш опыт работы позволяет разобраться с этим, а интерес к экспериментам выше желания повторять готовые решения, то эксперименты с включением сенсорных панелей в систему могут оказаться

очень увлекательными. При этом желательно, но не обязательно, проводить их с панелью, связанной с системой интерфейсом RS485. В этом случае вы можете использовать все рекомендации, описанные в книге.

Вообще вопрос использования в самодельной системе устройств других систем связан с протоколом их работы. Здесь все зависит от вашего опыта, знаний, желания разобраться во всем и, конечно, финансовых возможностей.

Применение в системе сенсорных панелей необязательно. Они не дают выигрыша в надежности, несколько расширяют возможности в отображении информации. Но и самые дешевые сенсорные панели, которые мне доводилось видеть, несомненно, украсят систему. Любая, даже очень сложная, система «Умный дом» в идеальном варианте не видна. Она спрятана в силовых или слаботочных шкафах, укрыта в мебели или встроенных в стены стойках. Единственное, что остается на виду, – устройства управления. В этом смысле современные сенсорные панели могут украсить комнату.

По ключевым словам «HomeAutomation» в поисковых системах Интернета можно найти много домашних страниц, посвященных созданию систем домашней автоматизации. Некоторые авторы приводят подробное описание своих проектов, схемы, программы, включая исходные коды.

Что-то из сделанного другими любителями может прийтись вам по душе. Или один из проектов увлечет вас. Например, есть описания управляющих программ, не уступающих профессиональным, а вы не стремитесь к созданию собственной программы. Или проект не имеет некоторых модулей, которые есть в вашей системе, а ориентирован только на купленные модули другой системы. Опять речь идет о смешанных системах. И все, что сказано выше, применимо к смешанным системам любительского диапазона.

Применение других любительских разработок открывает значительно больше возможностей, чем использование купленных устройств. Я имею в виду, что любители всегда найдут возможность поддержать друг друга, помочь друг другу.

Кто-то хорошо разбирается в программировании, кому-то легче работать с аппаратурой. Совместными усилиями вы быстрее и лучше выполните любой проект. В этом смысле смешанные системы в любительской практике, – скорее, правило, чем исключение.

Разные подходы к реализации системы

Говоря о смешанных системах, я упоминал другие системы, но не рассказывал о них. Приведу фрагмент статьи, восполняющий этот пробел.

Вначале – коммуникации в системе автоматизации дома.

Для создания работающей системы автоматизации дома мы должны расширить управление оборудованием до конкретного устройства, конкретного места или уровня замкнутой цепи. Чтобы сделать это, необходима некоторая точка сосредоточения, осуществляющая сетевое взаимодействие и позволяющая командам и данным перемещаться между системными компонентами.

Команды должны иметь возможность достигать оборудования везде в доме, где бы оно ни располагалось. Наиболее развитые системы автоматизации дома осуществляют двухстороннюю связь, допуская обратную связь от управляемого устройства к контроллеру или программе.

Есть несколько схем системной коммуникации, каждая со своими достоинствами и недостатками. Самые лучшие системы автоматизации дома используют системный центр, который объединяет несколько типов соединений, зависящих от выбранного оборудования и его использования. Наиболее общие типы системных центров включают:

- типовую проводку в доме;
- несущие силовых линий;
- коаксиальный кабель;
- специализированные соединители витых пар;
- низковольтную проводку;
- беспроводные радиочастотные сигналы;

- беспроводные инфракрасные сигналы;
- оптический кабель.

Если вы хотите использовать типовую проводку дома в качестве системной основы, ваши возможности будут несколько ограничены, но все же в вашем распоряжении останется достаточно много функций, предназначенных для автоматизации. Добавив модули контроллеров со специальными возможностями коммуникаций по силовым проводам, или используя беспроводные связи, вы сможете интегрировать значительное количество оборудования в автоматизированную систему дома со значительным объемом централизованного управления.

Но что вы сможете сделать без использования этих возможностей? Без изменения имеющейся проводки? Обычно управление оборудованием и автоматизация ограничены в возможностях управлением, которое было включено в оборудование. Например, телевизор и видеомагнитофон могут управляться с переносного пульта, микроволновые печи имеют встроенные возможности программирования, а система полива имеет свой собственный программируемый контроллер. Вы можете также заменить старые «глупые» устройства такими же, но «разумными». Так, термостат с ручной установкой может быть заменен программируемым термостатом, а выключатель с детектором движения может сменить старый ручной выключатель. Некоторые из этих разумных устройств могут поставляться с приспособлениями беспроводного управления, хотя все это, в основном, работает только с конкретным устройством или несколькими подобными.

Принципиальным ограничением стандартной проводки является то, что без дополнительных коммуникационных возможностей управление ограничено конкретными устройствами. Ограничен эффективный радиус действия удаленного управления. Например, вы можете использовать переносной пульт управления для управления или программирования видеомагнитофона, но он работает, только если видеомагнитофон расположен так, что может получить инфракрасный

сигнал от пульта – вы не можете управлять магнитофоном из другой комнаты или с другого этажа.

Система X10 использует стандартизованный протокол и множество команд, поддерживающих множество задач автоматизации. Она реализуется на коммуникациях силовых линий, накладывая сигнал с высокочастотной несущей (120 кГц) поверх 60 Гц (или 50 Гц) несущей силовой линии.

X10 контроллеры могут включаться в любом месте домашней проводки. Команды от контроллеров передаются через внутреннюю питающую сеть и не ограничены какой-либо штатной длиной цепи (межфазовые мосты позволяют сигналам переходить с фазы на фазу обычной трехфазной сети). Специальные блокирующие устройства могут быть установлены в силовом шкафу для ограничения перемещения сигналов X10 вне системного пространства, в другие дома или помещения с той же силовой цепью.

Любые команды посылаемые контроллерами X10 по домашней проводке будут получены всеми X–10-приемниками или модулями в этой системе. Некоторые модули могут не только принимать, но и отправлять сигналы.

Когда модуль получает команду, он проверяет «адрес», добавляемый к команде, – не ему ли она адресована. Если команда предназначена другому модулю, команда игнорируется. Когда соответствующий модуль получает команду, он действует согласно ей, включая или выключая что-либо, уменьшая яркость света, пересылая команду другому устройству и т.д.

Команды X10 ограничены множеством из 16 команд, называемым *стандартным функциональным множеством.* Однако большинство модулей распознает только базовое функциональное множество из семи команд.

Были также развиты расширенные пакеты кодов для увеличения функциональности при передаче, полезные для системы автоматизации дома при управлении такими подсистемами, как управление климатом, охранная, аудио и т.д. Однако из-за природы технологии X10 это снижает быстродействие, увеличивая количество циклов питающей сети, используемых для передачи дополнительной информации.

Коды адресации описываются 4-битовым кодом «дома» (буквы A–P) и 4-битовым кодом «устройства» (номера 1–16) – A-1, A-7, C-14, G-9 и т.д.

Для активизации одного или более модулей на выполнение функций пакеты сообщений с кодом «дом/устройство» передаются всем модулям, которым команды предназначены. Это «оклик» модулей. Затем отправляется другой пакет сообщений с описанием кода функции. Выбранные модули выполняют эти функции. Модуль «освобождается» после прихода кода функции, если первое полученное сообщение с кодом «дом/устройство» адресуется не ему, или если это функция – «все выключить».

Два наиболее общих типов модулей X10 – модули управления освещением и управления бытовыми электроприборами. Модули управления освещением обычно управляют активной нагрузкой, например лампами накаливания, и могут включать и выключать их или регулировать яркость свечения. Модули управления бытовыми приборами используют реле для включения и выключения присоединенных к ним устройств.

Контроллеры X10 могут быть отдельными подключаемыми устройствами или соединяться проводами в систему. Некоторые из них просты, только с несколькими основными функциями управления, тогда как другие могут содержать встроенные таймеры, часы и возможности доступа к телефону. Контроллеры обычно имеют встроенные клавиши, циферблаты и т.д., чтобы осуществлять пользовательский интерфейс с системой.

Другие устройства, добавленные для улучшения системы X10, – это радиочастотные (RF) и инфракрасные (IR) контроллеры и модули. Они используются как вспомогательные в ситуациях, не перекрываемых технологией передачи по силовым проводам. Например, охранная система может задействовать радиочастотные датчики для дверей и окон. Контроллер «базовая станция» получает радиокоманду от датчика, переводит ее в команду X10 и отсылает в силовую сеть. Естественно, расширение полезности системы X10 подобным

образом влечет за собой ее удорожание и увеличивает сложность без какого-либо улучшения в части надежности.

В самых больших системах могут потребоваться повторители, когда ослабление сигналов становится проблемой.

Система X10 достаточно дешевая и легко устанавливаемая, поскольку большинство ее компонентов используют имеющуюся в доме проводку как среду связи. Однако есть некоторые неудобства:

- для силовых линий характерен «шум», уменьшающий возможности передачи;
- топология проводки часто неизвестна, особенно для старых проводок;
- ослабление сигналов в силовой сети непостоянно и часто непредсказуемо, что ограничивает возможности передачи;
- другие технологии могут одновременно использовать силовые линии для коммуникации и взаимодействовать с коммуникациями X10;
- более ограниченное множество команд, чем у других технологий;
- скорость передачи данных низкая, ограничивается применением в тех случаях, когда время не критично, и неважна передача больших объемов данных;
- требует, по меньшей мере, одного передатчика (контроллера) для передачи или «ввода» сигнала в силовую проводку, а также отдельный приемник (системный интерфейс) для каждого управляемого оборудования или цепи;
- некоторые плохо продуманные устройства X10 могут иметь коллизии при совместных передачах.

Новые улучшения в технологии X10 увеличивают возможности системы, но и являются источником проблем. Исполнение этой технологии довольно трудно поддается усовершенствованию – требует дополнительных затрат и сложности, что противоречит назначению системы X10: дешевой системы, которая одинаково хороша для большинства нужд домашней автоматизации.

Новые модули X10 все время развиваются для расширения круга задач. Как правило, они могут быть передатчиками, приемниками или приемопередатчиками, могут разрабатываться для управления специфическими типами оборудования. Итак, что может делать модуль? Вот неполный список:

- переключение – включать и выключать сетевые устройства, такие как лампы;
- регулировка уровня/скорости – непосредственно регулировать яркость света и управлять скоростью двигателей;
- теле-/аудио-/видеоконтроллеры – посылают специальные команды другим устройствам: телевизорам, видеомагнитофонам, радиоприемникам, аудиоустройствам и т.д.;
- X–10-совместимый термостат может показывать температуру;
- датчик освещенности, датчик температуры, датчик ветра;
- управление открыванием/закрыванием;
- автоответчик и т.д.

Некоторые ведущие производители оборудования X10 – Honeywell, Leviton, Advanced Control Technologies, IBM, и X10 USA.

Стандарт CEBus позволяет использовать множество сетевых сред, включая:

- витые пары;
- коаксиальный кабель;
- инфракрасные сигналы;
- радиочастотные сигналы;
- оптоволоконные сигналы;
- сигналы по силовым сетям;
- аудио/видеошины.

Коммуникационное оборудование CEBus, язык и протокол реализованы на микросхеме, сделанной фирмой Intellon Corporation, Ocala, Florida. Intellon продает микросхему

производителям для встраивания в их разработки. Intellon может также производить специальные OEM CEBus продукты и средства разработки.

Системы автоматизации дома CEBus могут устанавливаться в жилом доме с использованием существующей проводки 110 В или 220 В для обмена данными. В случаях, когда это не оптимально для передачи данных, например в случае видео сигналов, предлагаются дополнительные возможности, особенно там, где проводная связь не слишком удобна или слишком дорога. Для целей удаленного управления обычно используются инфракрасные или радиоканалы. Технология CEBus может встраиваться в конкретные бытовые приборы или устройства удаленного управления, включаемые в сетевую розетку.

Стандарт CEBus включает такие технологии, как спектрально модулированный поток в силовых линиях. Спектральный поток начинает модуляцию на одной частоте и изменяет ее за время цикла. Начиная сигнал с частоты 100 кГц, его частоту линейно увеличивают до 400 кГц за период 100 мкс. Сигнал («высокое» состояние) и отсутствие сигнала («низкое» состояние) могут воспроизвести простейшие цифры, поэтому пауза между ними не требуется.

Логическая «1» может воспроизводиться отсутствием (или наличием) сигнала в течение 100 мкс. Логический «0» воспроизводится отсутствием (или наличием) сигнала в течение 200 мкс. Это означает, что длительность передачи переменная и зависит от количества «1» или «0».

Вне зависимости от используемой среды в CEBus управляющие данные канала транслируются со скоростью 8000 бит в секунду. Среда также может нести канальные данные. Пропускная способность будет зависеть от возможностей среды. Управляющие сообщения CEBus всегда имеют один формат. Сообщение содержит адрес получателя, но не имеет информации о маршруте, так что получатель может быть где угодно в сети, в любой среде.

Управляющие каналы CEBus несут сообщения о статусе и команды. Эти сообщения состоят из строк или пакетов байт

данных, которые могут иметь переменную длину, зависящую от количества данных в сообщении. Пакеты могут быть в сотни бит длиной. Минимальный размер пакета – 64 бита. Он занимает промежуток времени 1/117 с на передачу и прием.

Есть команды для назначения каналов данных, но в значительной мере стандарт CEBus заботится о спецификации управляющих каналов. Включение каналов данных не специфицировано в стандарте CEBus.

Адрес устройств устанавливается аппаратно при производстве и допускает 4 млрд. комбинаций. CEBus имеет также определения языка объектно-ориентированного управления, включающего такие команды, как увеличить уровень, быстро вперед, перемотать, пауза, пропустить, температуру поднять или опустить на один градус и т.д.

Сообщения могут быть адресованы (посланы) конкретному устройству. Может использоваться и специальная адресация для обращения ко всем устройствам сети или некоторой группе устройств. Адреса бывают индивидуальными или групповыми. Устройства могут принадлежать более чем к одной группе. Заметьте: не все CEBus совместимые устройства поддерживают групповую адресацию. Это определяется производителем и моделью.

Технология CEBus позволяет устройствам располагаться в любом месте сети на любом расстоянии, независимо от среды, если они имеют соответствующий для нее CEBus-интерфейс. Сообщения, посылаемые из среды одного типа в другую, проходят через маршрутизирующую цепь. Маршрутизатор может быть отдельным устройством или интегрироваться в бытовой прибор.

Управляющие сообщения распределяются между CEBus-устройствами и маршрутизаторами. Централизованный контроллер не применяется для управления доставкой. Никакая специальная топология в CEBus не стандартизируется. Все точки соединения управляемых устройств в среде обрабатываются логически так, как если бы они были на той же шине. Это означает, что все управляемые устройства в каждой среде чувствуют поступление пакетов данных одновременно. Все

устройства прочитывают адрес получателя в сообщения, но только те, кому они адресованы, получают остальную часть сообщения и отвечают на нее.

Соблюдаются требования по аудио/видео, но спецификации совершенно не были реализованы в EIA. Они описывают многоконтроллерную шину, состоящую из восьми витых пар. Она разработана для внутреннего соединения группы домашнего оборудования в небольшом пространстве (в одной комнате). Максимальная длина кабеля – 30 футов. Кабель несет три аудиоканала, четыре видеоканала и управляющий канал. Единственный разъем используется для подключения кабеля к управляемому устройству.

В отношении коаксиального кабеля CEBus определяет двухкабельную систему. Один кабель присоединяется к источникам сигнала в доме (камеры, магнитофоны и т.д.), а другой используется для распространения сигнала к любому приемнику в системе. Любые внешние видеосигналы могут комбинироваться с сигналами от источников в доме.

Система Echelon базируется на наличии разумно управляемых устройств (узлов). В системе Echelon, также называемой LonWorks-сетью, соединение между устройствами может быть либо один–к–одному (распределенное управление), либо ведущий–ведомый (централизованный контроль). Общий протокол используется в обоих случаях.

Какой бы вариант ни использовался (централизованный или распределенный), каждый узел имеет высокий уровень встроенной разумности. Компьютерные возможности узлов допускают распределение функций процесса по всей системе. Например, температурный датчик на Echelon технологии может быть предварительно запрограммирован для анализа прочитанной локальной температуры и фильтрации результатов, так что к центральному контроллеру или другому узлу будут направляться только определенные изменения. Функции управления тоже могут распределяться по всей системе, обеспечивая наилучшую производительность и надежность.

Узлы соединяются один с другим на основе равноправия, используя общий протокол. Каждый узел содержит внутреннюю разумность, поддерживающую протокол, управляющие

функции и функции процесса, а также включает интерфейс ввода-вывода, который присоединяет микроконтроллер узла к коммуникационной сети. В каждом узле большинство этих возможностей реализовано на одной микросхеме, называемой Нейрон-чип, и производимой в нескольких версиях фирмами Motorola и Toshiba. Технология Echelon LonWorks является открытой, но патентованной.

Подход Echelon не предусматривает использования имеющейся проводки для коммуникации. Соответственно, требует дополнительной проводки при установке, что избавляет от проблем и ограничений, связанных с применением силовой проводки, как в системах подобных X10. Передача идет много быстрее и значительно надежнее, не требует интерфейса подключения к силовой сети, не зависит от постоянных или меняющихся условий в проводке. Вдобавок, Echelon не ограничено никакими исключительными типами сетевых коммуникационных проводок/соединений. Допустимо множество передающих и соединительных устройств. Поддерживается вход в Ethernet, T1, X.25, Bitbus, Profibus, CAN, Modnet, SINEC, Grayhill, Opto22 (digital), OptoMux, Modbus, ISA bus, STD32 bus, PC/104 bus, VME bus и EXM bus.

Встроенные возможности Нейрон-чипа предлагают широкий круг вычислительных и разумных функций, осуществляемых на локальном уровне, но технология Echelon LonWorks не ограничена процессами узлового уровня, она поддерживают также возможность работы с хост-приложениями, которые работают на иных процессорах, чем любой из Нейрон-чипов. Должен быть компонент микропроцессора, который использует Нейрон-чип в качестве коммуникационного сопроцессора или Windows-ориентированный персональный компьютер, который связан с системой через серийный порт или адаптер PC. Эти приложения позволяют добавить в систему серверы, консоли или мониторы, и могут проводить централизованное управление для не распределенных систем автоматизации. Они также предназначены для переноса имеющихся приложений, работы утилит сложного сетевого управления и создания входов в другие системы.

Типичный узел в системе Echelon предназначен для простых задач. Такие устройства, как датчики приближения, переключатели, диммеры, датчики движения, реле, контроллеры моторов, шаговые двигатели и т.п., могут быть узлами сети. Объединенное множество коммуникационных узлов создает супермножество сложных функций управления, требуемых для автоматизации дома.

В системе Smart House управляющие сигналы для бытовой техники, информация о состоянии, данные сообщений приходят по одному каналу и используют тот же самый протокол. Исключение составляют видеосигналы и телефонные данные.

Базовая топология состоит из звезды с ветвями, относящимися к электрическим ответвлениям цепей. В концентраторе звезды находится системный контроллер, который обычно располагается в непосредственной близости от сетевых выключателей электросистемы дома. Системный контроллер включает бесперебойное питание, экранирование от наводок, GFI, телефонный ввод и наконечник для коаксиального кабеля.

Системный контроллер выполняет следующие задачи:

- обслуживание системы;
- пересылку данных бытовым приборам;
- планирование обслуживания.

Все пересылаемые сообщения поддерживаются системным контроллером. Smart House имеет специализированный формальный язык для управления приборами и сообщений о состоянии.

Система Smart House базируется на трех группах кабелей или типах:

- ответвляющих кабелях – силовых и цифровых; они состоят из силового кабеля и отдельного цифрового кабеля (четырехпарного, витого); одна витая пара – для данных, другая – для передачи сигнала синхронизации, а оставшиеся две не используются или могут быть использованы для отдельных ветвей;

- прикладных кабелей – постоянное напряжение для датчиков и цифровых данных;
- коммуникационных кабелей – телефонные провода и коаксиальный видео кабель.

Проводка Smart House наиболее подходит для новых разработок, поскольку предотвращает необходимость в последующих переделках.

Настенные выключатели не соединены непосредственно с розетками. Они соединяются прикладными кабелями, что включает коммуникационные каналы. Когда выключатель разомкнут или замкнут, сигнал поступает к системному контроллеру, который действует в соответствии с программируемыми таблицами для управления различными лампами или розетками. Такой подход позволяет осуществлять перераспределение приборов и переназначение выключателей, при котором тот же самый выключатель может выполнять разные функции, зависящие от времени дня или других факторов.

В системе Smart House определена двойная система коаксиальных кабелей. Снижающие кабели несут видеосигналы от источников – кабельного телевидения, спутниковых тарелок, антенн и т.д. Поднимающие кабели несут видеосигналы от внутренних источников – видеомагнитофонов, компьютерных терминалов и т.п. Наконечники сервисного центра Smart House комбинируют эти поднимающиеся и снижающиеся сигналы для распределения по снижающим коаксиальным кабелям. Этот подход аналогичен системе CEBus.

Телефонный вход осуществляет функции интеркома и доступа системы Smart House для операций с удаленным управлением. Специфическое звучание звонка отличает вызов по интеркому от звонков остальных телефонов в доме. Телефонный вход включает модем для целей удаленного управления. Тональные сигналы с внешнего телефона могут активизировать программируемые посылки данных для действующих приборов или сервисов в доме. Доступ закрыт паролем. Инструкции синтезированным голосом поддерживают удаленный контроль.

Приложение

В эту часть книги я включил некоторые справочные материалы.

Многие из описаний, в частности, описание самого контроллера, можно в переводе на русский язык найти на сайте российского представительства производителя микросхемы.

ИК-датчик движения

Схема датчика взята из Интернета. Приведена схема, скорее, для ознакомления, чем для повторения (рис. П.1).

Пироэлектрический сенсор изготовлен из кристаллического материала, который генерирует поверхностный электрический заряд, когда подвергается нагреву ИК-излучением. Изменяющееся количество заряда может быть измерено чувствительным FET-устройством, встроенным в сенсор. Элементы сенсора чувствительны к излучению в широком диапазоне, так что в корпус TO5 добавлен светофильтр для ограничения проникающего излучения диапазоном от 8 до 14 мкм, который наиболее подходит для обнаружения излучения от тела человека.

Как правило, вывод 2 FET соединен через отводящий резистор примерно в 100 кОм с общим проводом. И приходит на двухполярный усилитель, имеющий цепь ограничения сигнала. Усилитель обычно бывает фильтрующим с частотой

среза 10 Гц для удаления высокочастотных шумов и сопровождается компаратором, реагирующим как на положительные, так и на отрицательные переходы выходного сигнала сенсора. Хорошо отфильтрованное напряжение питания от 3 до 15 В должно быть присоединено к выводу 1 FET.

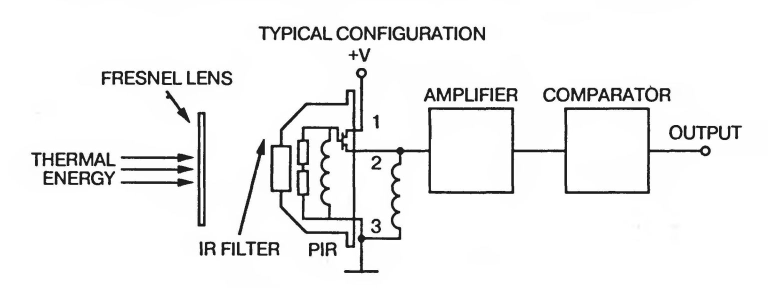


Рис. П.1. Структурная схема датчика движения

Сенсор PIR325 имеет два чувствительных элемента, объединенных в вольтокомпенсирующую конфигурацию. Это предотвращает воздействие сигналов при вибрации, изменении температуры и засветке. Человек, проходя перед сенсором, активирует первый элемент, а затем – второй, тогда как другие факторы воздействуют на оба элемента сразу и компенсируются ими. Источник излучения должен пересекать сенсор в горизонтальном направлении, когда выводы 1 и 2 расположены горизонтально, так что элементы последовательно подвергаются воздействию ИК-излучения. Перед сенсором обычно устанавливается фокусирующее устройство.

Типовое применение – в схеме с исполняющим реле (рис. П.2). Резистор R10 и конденсатор C6 изменяют время, в течение которого реле RY1 активизировано после обнаружения движения.

В качестве фокусирующего устройства используются линзы Френеля.

Линзы Френеля – это выпуклые в разрезе линзы, сжатые в себя так, чтобы принять форму плоской линзы с сохранением оптических характеристик, но меньшей толщины. Благодаря этому они менее подвержены абсорбционным потерям.

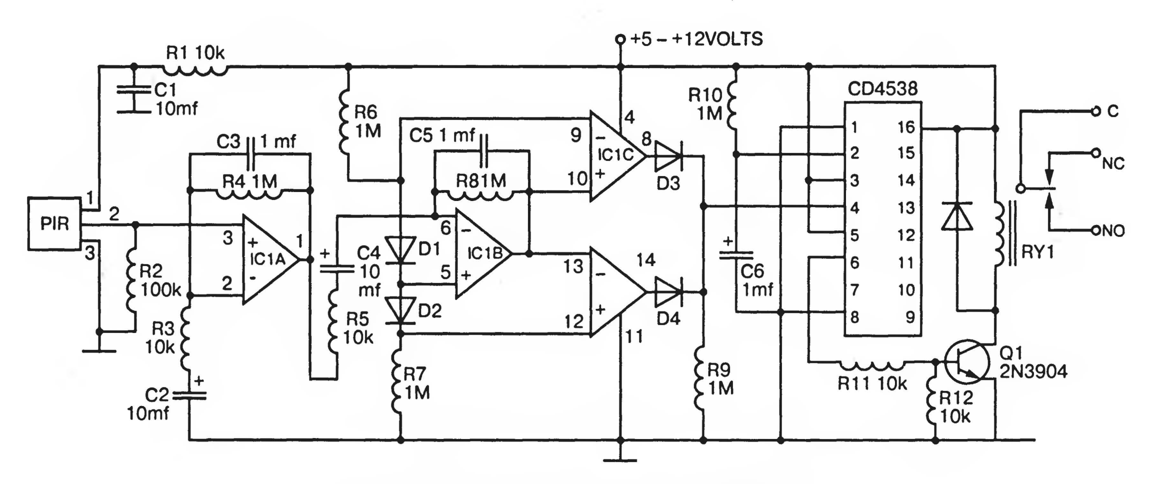


Рис. П.2. Принципиальная схема датчика движения

Таблица команд микроконтроллера PIC16F628A

Таблица П.1. Описание полей кодов операций PIC16F628A

Поле	Описание
f	Адрес регистра (от 0×00 до 0×7F)
W	Рабочий регистр (аккумулятор)
b	Бит, адресуемый внутри 8-битового регистра
k	Константа или метка
x	Не обслуживаемое расположение (= 0 или 1). Ассемблер будет генерировать код с x = 0. Это рекомендованная форма использования для совместимости со всеми программными средствами Microchip
d	Выбор адресата: d = 0, сохранить результат в W; d = 1, сохранить результат в регистре f. По умолчанию d = 1
label	Имя метки
TOS	Top of Stack (Верх стека)
PC	Program Counter (Счетчик команд)
PCLATH	Program Counter High Latch (Защелка старшего байта счетчика команд)
GIE	Global Interrupt Enable bit (Бит глобального разрешения прерываний)
WDT	Watchdog Timer/Counter (Watchdog Таймер/Счетчик)
TO	Time out bit (Бит окончания по времени)
PD	Power-down bit (Бит сброса по питанию)
dest	Адресуется либо регистр W, либо определенный положением регистр
[ ]	Опции
( )	Содержимое
→	Назначить
< >	Поле бита регистра
∈	Во множестве
italics	Пользовательский термин (шрифт courier)

Таблица П.2. Основные команды контроллера PIC16F628A

Мнемоника, операнды	Описание	Циклы	14-битовый код	Статус, касается	Примечания
Байт-ориентированные регистровые операции					
ADDWF f, d	Сложить W и f	1	00 0111 dfff ffff	C,DC,Z	1,2
ANDWF f, d	«И» W c f	1	00 0101 dfff ffff	Z	1,2
CLRF f	Очистить f	1	00 0001 1fff ffff	Z	2
CLRW -	Очистить W	1	00 0001 0000 0011	Z	
COMF f, d	Дополнение f (двоичное – инвертировать все цифры и добавить 1)	1	00 1001 dfff ffff	Z	1,2
DECF f, d	Декремент f	1	00 0011 dfff ffff	Z	1,2
DECFSZ f, d	Декремент f, Пропустить, если 0	1(2)	00 1011 dfff ffff		1,2,3
INCF f, d	Инкремент f	1	00 1010 dfff ffff	Z	1,2
INCFSZ f, d	Инкремент f, Пропустить, если 0	1(2)	00 1111 dfff ffff		1,2,3
IORWF f, d	Включающее «ИЛИ» W c f	1	00 0100 dfff ffff	Z	1,2
MOVF f, d	Переместить f	1	00 1000 dfff ffff	Z	1,2
MOVWF f	Переместить W в f	1	00 0000 1fff ffff		
NOP -	Нет операции	1	00 0000 0xx0 0000		
RLF f, d	Сдвинуть влево через перенос	1	00 1101 dfff ffff	C	1,2
RRF f, d	Сдвинуть вправо через перенос	1	00 1100 dfff ffff	C	1,2
SUBWF f, d	Вычесть W из f	1	00 0010 dfff ffff	C, DC, Z	1,2
SWAPF f, d	Поменять местами полубайты f	1	00 1110 dfff ffff		1,2
XORWF f, d	Исключающее «ИЛИ» W c f	1	00 0110 dfff ffff	Z	1,2

Бит-ориентированные регистровые операции					
BCF f, b	Очистить бит f	1	01 00bb bfff ffff		1,2
BSF f, b	Установить бит f	1	01 01bb bfff ffff		1,2
BTFSC f, b	Проверить бит f, пропустить, если сброшен	1(2)	01 10bb bfff ffff		3
BTFSS f, b	Проверит бит f, пропустить, если установлен	1(2)	01 11bb bfff ffff		3
Операции управления и с константами					
ADDLW k	Сложить константу и W	1	11 111x kkkk kkkk	C,DC,Z	
ANDLW k	«И» константы с W	1	11 1001 kkkk kkkk	Z	
CALL k	Вызвать подпрограмму	2	10 0kkk kkkk kkkk	TO,PD	
CLRWDT -	Очистить сторожевой таймер (Watchdog)	1	00 0000 0110 0100		
GOTO k	Перейти к адресу	2	10 1kkk kkkk kkkk		
IORLW k	Включающее «ИЛИ» константы с W	1	11 1000 kkkk kkkk	Z	
MOVLW k	Переместить константу в W	1	11 00xx kkkk kkkk		
RETFIE -	Возврат из прерывания	2	00 0000 0000 1001		
RETLW k	Возврат с константой в W	2	11 01xx kkkk kkkk		
RETURN -	Возврат из подпрограммы	2	00 0000 0000 1000	TO, PD	
SLEEP -	Переход в режим Standby	1	00 0000 0110 0011		
SUBLW k	Вычесть W из константы	1	11 110x kkkk kkkk	C, DC, Z	
XORLW k	Исключающее «ИЛИ» константы с W	1	11 1010 kkkk kkkk	Z	

Примечания.

1. Когда модифицируется регистр I/O как функция сама по себе (то есть, MOVF PORTB, 1), используемое значение будет то же, что значение на самом выводе. Например, если данные на выводе, конфигурированном как ввод «1», сбрасываются внешним устройством, возвращаемые данные – это «0».
2. Если эта инструкция выполняется на регистре TMR0 (и применено d = 1), предделитель будет очищен, если назначен для Timer0 Module.
3. Если счетчик команд (PC) модифицируется или проверяемое условие истинно, инструкция требует двух циклов. Второй цикл выполняется как NOP.

Цоколевка контроллера PIC16F628A

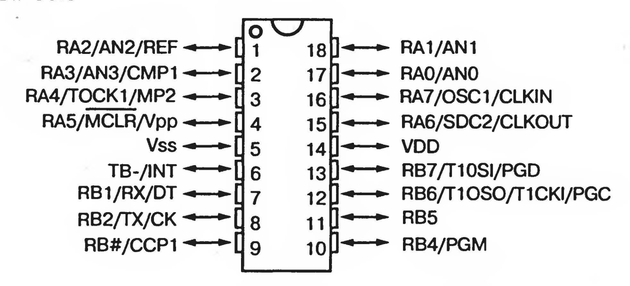


Рис. П.3. Цоколевка контроллера PIC16F628A

Программатор (совместно с PonyProg)

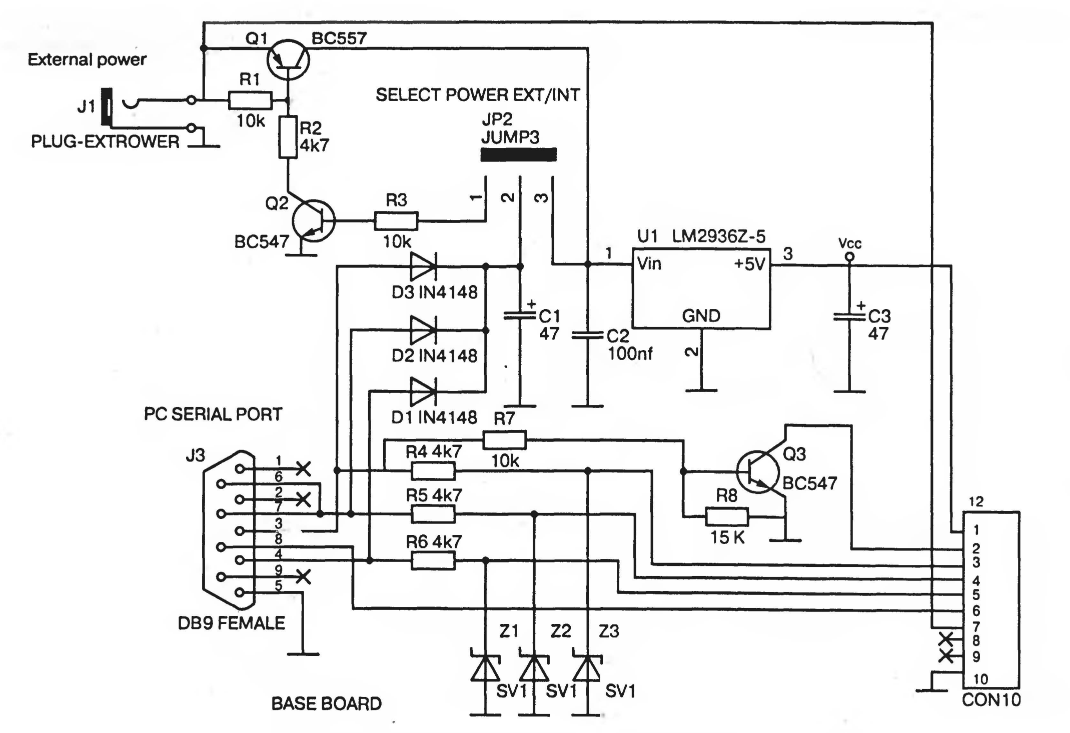


Рис. П.4. Схема программатора для контроллеров PIC

Адаптер для PIC-контроллеров

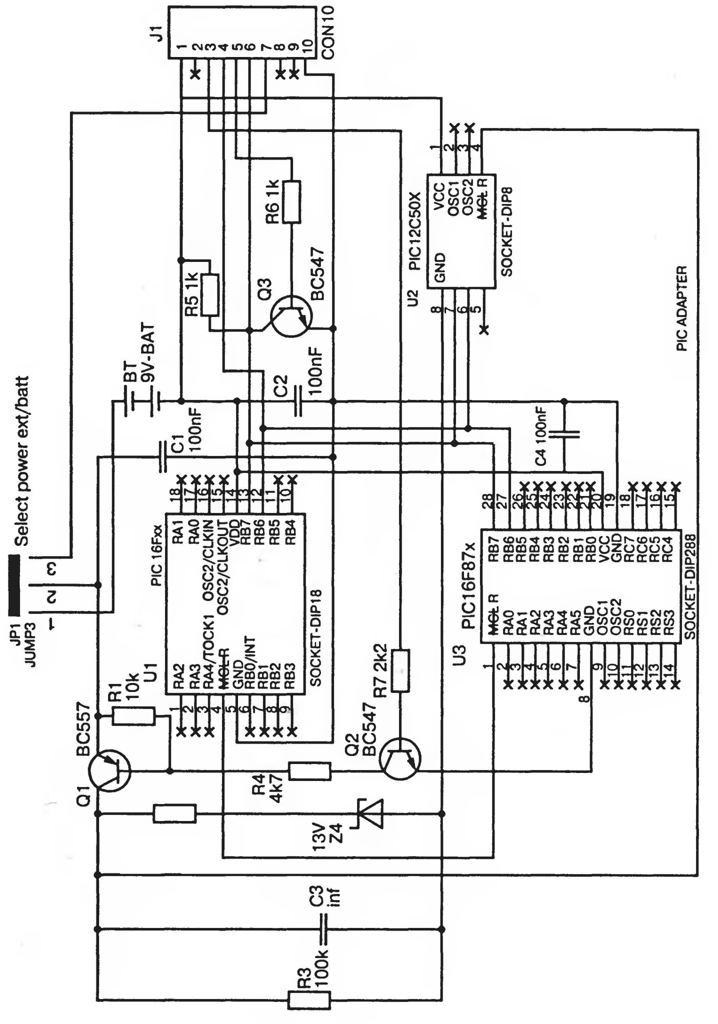


Рис. П.5. Схема адаптера к программатору

Внешний вид и параметры модуля общего назначения фирмы Advantech

ADAM 4060. Модуль релейного цифрового вывода (рис. П.6):

- 2 релейных выхода типа A (двухпозиционный вывод);
- 2 релейных выхода типа C (трехпозиционный вывод);



Рис. П.6. Внешний вид модуля ADAM

Параметры контактов реле по переменному току:
125 В @ 0,6 А, 250 В@ 0,3 А,
по постоянному току:
30 В @ 2 А, 110 В @ 0,6 А
Интерфейс RS485

Практическое применение триака в модулях системы

Хотя на протяжении всей книги я старался подчеркнуть, что все модули системы, как и сама система, предназначены для проведения экспериментов за компьютером или за столом, думаю, среди читателей найдутся желающие воплотить в жизнь отдельные модули или на базе предложенных модулей создать свою систему. Для тех, кто знаком с техникой безопасности и может работать с напряжением ~220 В, в справочной части книги я хочу привести несколько практических советов по применению триака в модуле управления светом.

Для этих целей вполне подойдет триак Q6025L6, X44081 или аналогичный. В реальных конструкциях, с которыми мне приходилось сталкиваться, вполне успешно работают триаки с допустимым напряжением 400 В в закрытом состоянии, но лучше использовать 600-вольтовый вариант, если есть возможность выбора.

Второе, о чем мне хотелось бы сказать, – это способ управления. Ниже я привожу реальную схему включения оптопары (рис. П.7). Включение диммера начинается с реле, контакты которого КР подключают нагрузку. Кроме тех соображений, которые мне не ведомы, подобное подключение обеспечивает большую электрическую безопасность. В Интернете есть статьи по тиристорному управлению, и, прежде чем принимать окончательное решение, я советую почитать рекомендации по выбору коммутатора. Или поискать эту информацию в литературе. Если вы не планируете регулировать яркость светильника с помощью триака, желательно включать его сразу после перехода сетевого напряжения через ноль (или в момент перехода через ноль). Если же вы собираетесь регулировать яркость, оптимальным, как мне кажется, было бы удовлетвориться несколькими уровнями яркости и не включать триак в моменты, когда сетевое напряжение близко к амплитудному.

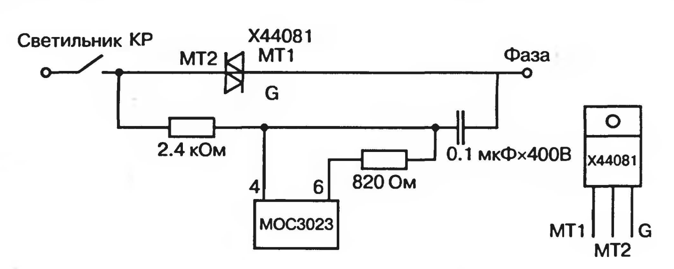


Рис. П.7. Схема управления триаком

Хотя оба используемые в схеме резистора могут иметь мощность 0,25 Вт, я советую составлять каждый из двух резисторов по 0,25 Вт. Например, резистор 2,4 кОм – из двух резисторов по 1,2 кОм. Это вызвано не необходимостью в увеличении мощности, а необходимостью увеличить допустимое

падение напряжения на резисторах, поскольку для резистора типа ОМЛТ при мощности 0,25 Вт допустимое напряжение – 250 В. В сети амплитуда напряжения может быть больше.

По возможности, лучше использовать триак в корпусе с изоляцией пластины крепления от вывода MT2 (они существуют в обоих исполнениях), если триак будет устанавливаться на теплоотвод, служащий одновременно и конструктивным элементом. Иначе теплоотвод может оказаться под высоким напряжением.

При выборе типа полупроводника по току следует учитывать, что лампы накаливания при включении имеют минимальное активное сопротивление, которое увеличивается по мере разогрева нити накала.

В любом случае, настоятельно рекомендую обратиться к информации по триакам (или тиристорам) до принятия решения о покупке деталей. Современные триаки 3Q вполне успешно работают не только на активную нагрузку, но и на реактивную, что позволяет применять их для регулирования скорости вращения двигателя, например вентилятора.

Очень интересное предложение по регулированию напряжения на инерционной нагрузке я встретил на одном из сайтов. Сущность идеи состоит в том, чтобы циклически пропускать несколько полуволн переменного напряжения. Скажем, для получения 70% яркости свечения лампы накаливания из десяти полуволн напряжения пропускается три, а семь приходят на нагрузку. Желательно их распределить равномернее по всему циклу. Выгода от подобного способа управления в том, что ключ открывается в моменты перехода напряжения через ноль, не создавая помех в сети, что характерно для фазового способа регулировки.

Дополнительные замечания по ИК-управлению

Программа WinLIRC позволяет не только считывать ИК-коды, но и воспроизводить прочитанные команды. Воспроизведение – чисто программное, возможно, по этой причине оно показалось мне не слишком уверенным. Но оно работает, что вы можете использовать в своих целях. На

сайтах, посвященных работе программы, неоднократно задавался вопрос о том, как следует изменить схему излучателя, чтобы модуль WinLIRC можно было отнести дальше от управляемого устройства. Простейшее решение – в применении транзисторного ключа, работающего на последовательно соединенные светодиод и токоограничительный резистор, который, в свою очередь, зашунтирован конденсатором. Величина емкости конденсатора зависит от параметров светодиода. Идея же заключается в том, что светодиод питается короткими импульсами большого тока. Любой светодиод допускает кратковременную подачу импульсов тока, во много раз превышающего средний допустимый. Скважность импульсов при этом зависит от конкретного типа прибора и может меняться от 100 для одних типов светодиодов до 3 для других.

В экспериментах, описанных в книге, я использовал светодиод АЛ307В, размещенный на макетной плате. При этом видеомагнитофон Sony управлялся с расстояния около 10 см. Для увеличения расстояния между модулем и управляемым устройством (если надо) можно, конечно, применить специализированный ИК-излучатель или воспользоваться приемом, который описан выше.

Обязательно ли использовать дополнительный транзисторный ключ? Я бы использовал его, как говорится, на всякий случай. Но, если вы уверены, что контроллер не будет выведен из строя, не используйте лишних элементов.

Если вместо микросхемы фотоприемника TSOP вы будете использовать фотоприемник другого типа, то можете столкнуться с явлением, которое однажды сильно озадачило меня. Использовался модуль фотоприемника, который транслировал полученные ИК-команды, для передачи команд из разных помещений на IR Xpander (системы X10). Последний кроме запоминания и воспроизведения ИК-кодов был способен распознавать коды, которые он предварительно запомнил. Эти коды впоследствии можно было использовать для инициирования любых сценариев в системе «Умный дом».

Первая проверка дала положительные результаты, но затем начались странные явления. IR Xpander то исправно выполнял команды, то не реагировал на них. При этом индикатор на панели показывал получение ИК-команд в моменты полного отсутствия ИК-излучения. Сделав неправильные предположения о причинах этого явления, я понизил напряжение питания фотоприемника, что дало положительные результаты. Только много позже я выяснил причины этого эффекта, разбираясь с другим устройством. Для увеличения чувствительности фотоприемника компаратор, стоящий после фотоэлемента, имел порог, очень близкий к уровню шумов. При неблагоприятных изменениях окружающей среды шумы пересекали порог компарации и транслировались как сигнал, который и заставлял IR Xpander надолго задуматься.

Эксперименты с ИК-командами весьма интересны и увлекательны, но не следует огорчаться, если что-то не получается. Я пробовал оценить, насколько уверено распознаются коды устройством, для этого предназначенным. Коды разных производителей распознавались им с достоверностью от 90–98% (ИК-коды одних производителей оборудования) до 5–10%.

Программа для компьютера в KDevelop

Причина, по которой я хочу рассказать о другой среде программирования, именно KDevelop, работающей с операционной системой Linux, – в ее большей доступности для многих, чем Visual Basic. Эта среда программирования входит в состав многих Linux-дистрибутивов. Хотя она и не единственная в них.

Итак, я предполагаю работать с релейным модулем. По этой причине я программирую микроконтроллер соответствующим образом и загружаю на компьютере ASPLinux. Я уже говорил, что эта операционная система поставляется с огромным количеством приложений, в частности, и для разработки программ.

Запускаем программу KDevelop. Моя версия в данный момент предоставляет в меню запуска возможность выбрать язык программирования и вариант проекта, хотя это можно сделать и несколько позже. Итак, я запускаю KDevelop в разделе основного меню **Приложения ➡ Разработка ➡ KDevelop KDE/C++**. Если это первый запуск, ни один проект не открывается, если повторный запуск, открывается последний проект, над которым вы работали. Для создания нового проекта предыдущий следует закрыть и выбрать в меню KDevelop **Проект ➡ Новый проект** (рис. П.8).

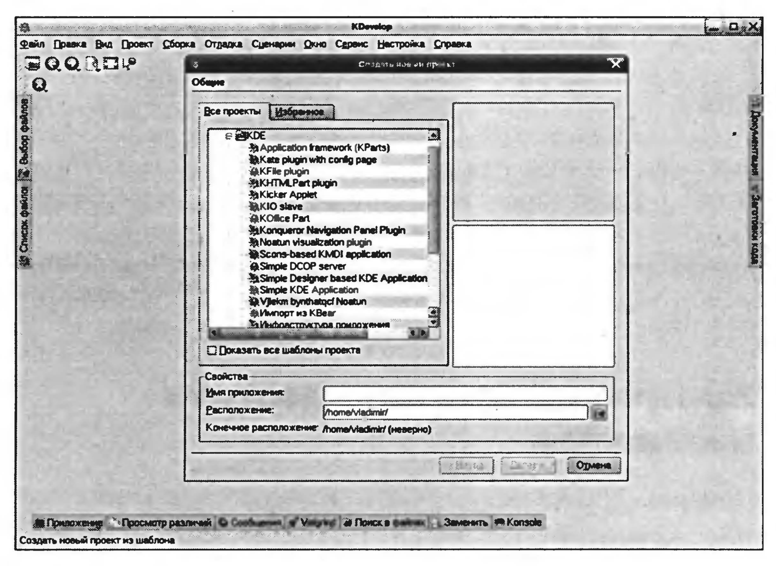


Рис. П.8. Создаем проект в KDevelop

В диалоговом окне выбираем **C++ ➡ KDE ➡ Simple Designer based KDE Application**, задаем имя проекта (в данном случае – book) и место его расположения (в данном случае – домашняя папка пользователя) – рис. П.9.

Теперь нам остается щелкнуть несколько раз кнопку **Next**, при этом можно сделать уточнения, относящиеся к проекту, но эти действия можно осуществить и позже. Заканчивается

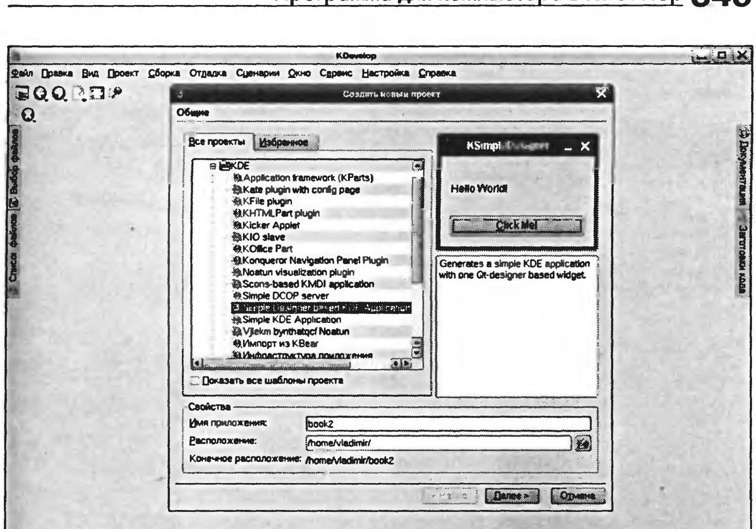


Рис. П.9. Выбираем тип проекта в KDevelop

оформление проекта, когда кнопка **Next** превращается в кнопку **Готово**, которую и щелкаем в последний раз, получая проект.

Щелкнув на правой инструментальной панели ярлычок **Automake Manager**, открываем окно менеджера, в котором можно открыть все файлы проекта и файлы заголовков двойным щелчком кнопки мышки. Я открыл файл bookwidgetbase.ui (рис. П.10).

Файл интересен тем, что относится к встроенному редактору интерфейса программы. Окно с кнопкой **Click Me!** появится после компиляции и сборки проекта при запуске программы. Для компиляции и сборки сначала в основном меню выбираем **Сборка ➡ Запустить automake и др...** В нижней части открывается окно сообщений, где показывается процесс сборки (рис. П.11).

Затем мне приходится сделать лишнее «телодвижение» – после одного из экспериментов в KDevelop с разными языками программирования мне приходится поправлять файл configure.sh, который находится в папке проекта book.

Рис. П.10. Встроенный графический редактор KDevelop

Рис. П.11. Окно процесса сборки проекта

Я открываю файл текстовым редактором, удаляю показанную часть текста почти в самом начале файла и сохраняю файл:

```
# Support unset when possible.
if ( (MAIL=60; unset MAIL) || exit) >/dev/null 2>&1;
then
as_unset=unset
else
as_unset=false
fi
```

Без этой операции появляются сообщения об ошибках при конфигурировании проекта, которое запускается в разделе основного меню **Сборка ➡ Запустить configure**. После завершения этого процесса можно запустить сборку **Сборка ➡ Собрать проект** или сразу запустить программу **Сборка ➡ Выполнить программу**, что приведет к сборке проекта и запуску программы.

Есть еще один момент, который привел меня в замешательство, но, возможно, это тоже результат моих экспериментов. Не получалась работа с графическим редактором интерфейса. Для его нормальной работы в опции проекта **Проект ➡ Project Options ➡ Поддержка C++** на вкладке **Автодополнение кода** в части **Code Complection Databases** понадобилось с помощью кнопки **Добавить** добавить поддержку Qt (рис. П.12).

После создания базы проблема перестала беспокоить. Qt, насколько я понимаю, – это мощная библиотека, поддерживающая графику. Разработана она фирмой Trolltech. Все графические объекты интерфейса – объекты этой библиотеки. Но вернемся к проекту.

Если теперь щелкнуть кнопку появившегося окна book **Click Me!**, появится традиционное для языка С – **Hello World!** Таким образом, почти незаметно для себя мы написали первую программу на языке С++, работающую с операционной системой Linux.

Теперь хотелось бы эту программу превратить в то, что нужно нам. Меня беспокоят некоторые аспекты предстоящей работы, но для начала я, выключив работающую

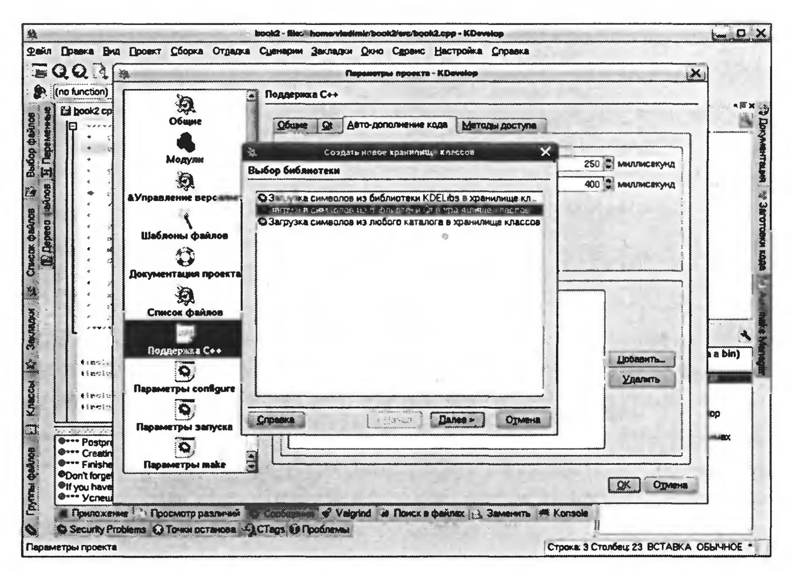


Рис. П.12. Добавление поддержки Qt

программу, возвращаюсь в редактор интерфейса к файлу bookwidgetbase.ui. С этой небольшой проблемой я столкнусь позже, но хочу устранить ее сейчас. Существо проблемы в том, что возникают некоторые неприятности с изменением интерфейса, а, изменив интерфейс, я не смогу справиться с размерами окна при запуске программы. Окно останется в том же первозданном виде, что и при первом запуске.

Основу интерфейса, как и в других средах программирования, составляет форма, на которой размещаются кнопки, этикетки и т.д. – все составляющие формы. Кроме нее открыта инструментальная панель (слева), менеджеры проекта, свойств формы и ее составляющих (справа).

Для устранения проблемы, вероятно, есть более правильные пути, но я пошел следующим – щелкнув правой кнопкой мыши форму, в раскрывающемся меню выбирал пункт **Разорвать расположение.**

Теперь я могу работать с формой. На форме в данный момент одна этикетка – **label**, и одна кнопка – **button**. Для начала я увеличу размер формы и поменяю местами компоненты, изменив их размер (рис. П.13).

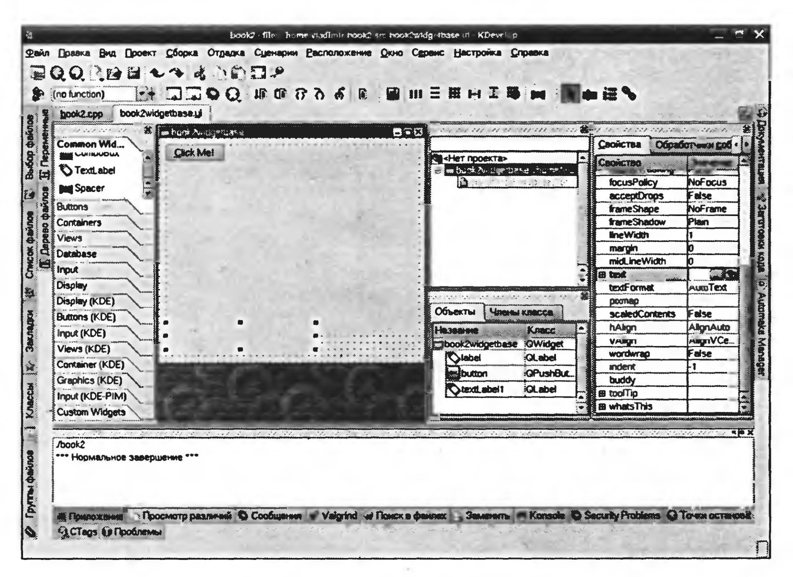


Рис. П.13. Работа с формой в KDevelop

Кнопку я позже использую для работы с портом, а этикетку – для вывода сообщений о состоянии порта. Добавим разделители, как показано на рис. П.14. Щелкнув правой кнопкой мыши форму, выбираем в раскрывающемся меню **Lay Out in a Grid**. Теперь при запуске программы окно приобретает вид, который бы мне хотелось видеть.

Однако вернемся к проблемам, вызывающим у меня беспокойство. Я бы сформулировал это так: «Плохо, когда забываешь то, что не знаешь. Особенно, если забываешь, что ты этого не знаешь!».

Производственная необходимость некогда заставила меня обратиться к языку C++, на котором я написал небольшую программку. Для знакомства с языком я законспектировал оригинальную версию книги Липпмана «Основы C++». С тех пор прошло достаточно времени, чтобы я все забыл. По этой причине я добавлю «шпаргалку», постаравшись не использовать сложные синтаксические или программные конструкции.

Первое, что я запишу, поскольку всегда путаюсь.

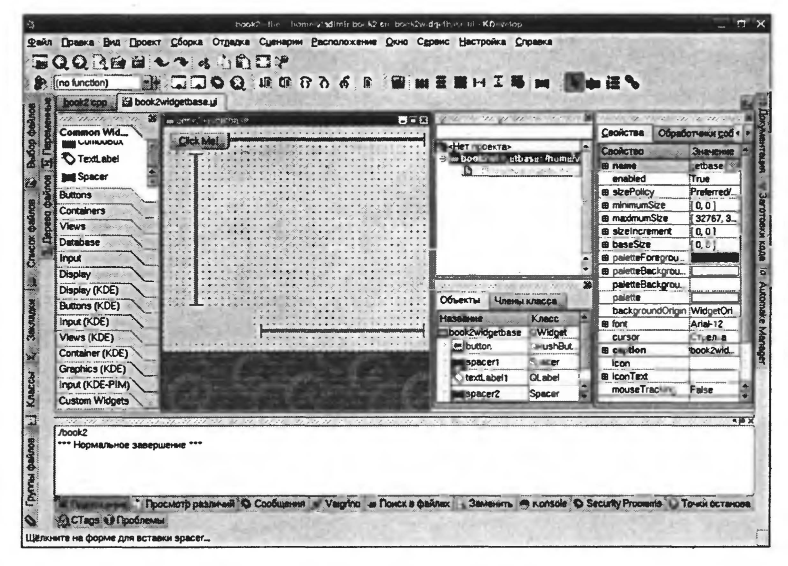


Рис. П.14. Придание форме нужного вида

Указатели в С.

Пусть мы имеем переменную с именем `var` *типа* `int` *(целое):* `int var;`

Теперь создадим другую переменную, которую назовем:

```
int* p_var.
```

Появившаяся звездочка говорит, что переменная `p_var` *– это указатель. Если теперь мы присвоим этой переменной следующее значение:*

```
p_var = &var,
```

то в переменной `p_var` *будет храниться адрес первой переменной. Положим, есть еще одна переменная:*

```
int new_var
```

Мы можем добавить к первой переменной единицу, присвоив результат новой переменной:

```
new_var = var + 1
```

Но в языке С то же можно записать, используя указатель:

```
new_var = *p_var + 1
```

звездочка в данном случае говорит о том, что мы используем значение переменной, адрес которой хранит переменная p_var.

Таким образом, &var *– указывает, что используется адрес переменной* var.

```
int* p_var
```

p_var *– это указатель,* *p_var *– показывает, что используется значение переменной, на которую указывает* p_var. *Указатели могут указывать на переменные, функции, другие указатели и т.д.*

Не знаю, поможет ли вам моя шпаргалка, но для себя я ее оставлю, тем более что в «С++» без звездочек, разбросанных по всему тексту, шагу не ступишь.

Еще одна деталь. Когда я впервые столкнулся с KDevelop, это была версия без интегрированного в нее редактора интерфейса. Для этих целей использовалась отдельная программа Qt Designer (она и сейчас входит в дистрибутив). И это не единственная возможность создать графический интерфейс, как и KDevelop не единственная среда разработки на языке С. Объединение программы на языке С++ и графического интерфейса выглядело несколько иначе, что заставляет меня, как мне кажется, сейчас делать много ненужного. Но я не собираюсь поразить воображение пользователей необычным дизайном программы, по причине чего не хочу пока разбираться с этим вопросом. В памяти осталось, что Qt Designer использует понятие слотов. В моем представлении, слот – нечто вроде гнезда, в которое вставляется модуль, как в материнскую плату вставляется модуль памяти. Используется и понятие сигналов, что для меня равносильно понятию использования выводов разъема. Через его выводы модуль получает сигналы от материнской платы или сам передает сигналы материнской плате.

И, действительно, открыв в KDevelop файлы bookwidget.cpp и bookwidget.h, в последней находим слот:

```
public slots:
   /*$PUBLIC_SLOTS$*/
   virtual void button_clicked();
```

А в первой находим сигнал, относящийся к этому слоту:

```
void bookWidget::button_clicked()
{
   if ( label->text().isEmpty() )
   {
      label->setText( "Hello World!" );
   }
   else
   {
      label->clear();
}
}
```

Этим для начала я и воспользуюсь, хотя, подозреваю, есть более простые способы реализовать то, что мне надо. В первую очередь я хочу переименовать кнопку из **button** в **port_open**. Я делаю это на вкладке bookwidgetbase.ui. В **Object Explorer** выбираю **button**, а в **Property Editor** в графе **name** меняю имя с **button** на **port_open**. Щелчком правой кнопки мыши открываю меню, выбираю **Слоты**, где меняю имя слота. Повторяю это, выбирая **Соединения**, где в разделе **слот** указываю новое имя. Затем меняю, чтобы не путаться в файлах .cpp и .h имя:

```
void bookWidget::port_open_clicked()
virtual void port_open_clicked();
```

Сохраняю эти файлы. Пытаюсь сохранить файл bookwidgetbase.ui, но он не хочет сохраняться. Тогда в выпадающем меню формы выбираю **Разорвать расположение**, сохраняю файл и повторяю **Lay Out in a Grid**, сохраняя файл.

В итоге проект собирается и запускается. В редакторе интерфейса я добавляю кнопки, которые называю **port_close** и **ButtonGroup** (и то, и другое из меню **Common Widgets** слева). Текст группы кнопок я меняю на **Гостиная** (двойной щелчок текста на форме открывает редактор текста). Пока не забыл, я вношу добавления в файлы bookwidget.cpp:

```
void bookWidget::port_close_clicked()
{};
```

и bookwidget.h:

```
virtual void port_close_clicked();
```

Затем открываю в редакторе интерфейса раздел **Слоты** и добавляю новый слот с помощью кнопки **Добавить функцию**:

```
port_close_clicked()
```

Отказываюсь генерировать новый класс. Добавляю новое соединение с помощью раздела **Соединения** и кнопки **Новое**, щелкая левой кнопкой мыши ячейки таблицы:

```
Sender - port_close
Сигнал - clicked()
Receiver -  bookwidgetbase
Слот - port_close_clicked()
```

И все сохраняю.

Для завершения этого этапа работы я добавляю еще две кнопки **Свет включить**, **Свет выключить** и из раздела инструментального меню **Display** этикетку **PixmapLabel**. Их я пока не оформляю. С PixmapLabel история простая – я попытался сделать включение и выключение ламп, как это делал в Visual Basic, но не получилось. Я не стал на этом «зацикливаться» и решил обойти проблему. В графическом редакторе KolourPaint я делаю полоску 350×30, которую закрашиваю в желтый цвет, а картинку сохраняю как light (рисунок **Portable Pixmap**) в папке проекта. Теперь картинку по умолчанию я поменяю на свой рисунок, который и будет изображать включенный свет (рис. П.15).

Под кнопкой **Свет выключить** на основной форме я расположил еще одну **TextLabel**, которую назвал **answer**. Ее я предполагаю использовать для получения ответа от релейного модуля для запроса статуса реле.

Вот, собственно, что я предполагал сделать в графическом плане. И хотя по дороге я несколько раз наступал на грабли, это были еще не «те» грабли.

С самого начала меня беспокоила и продолжает беспокоить сейчас возможность работы с COM-портом. Среда программирования в явном виде не предоставляет мне такой возможности, как это было в Visual Basic. Первое, что приходит в голову, – поискать в Интернете. И не напрасно. Я нахожу пакет qextserialport –0.9.0, автор которого:

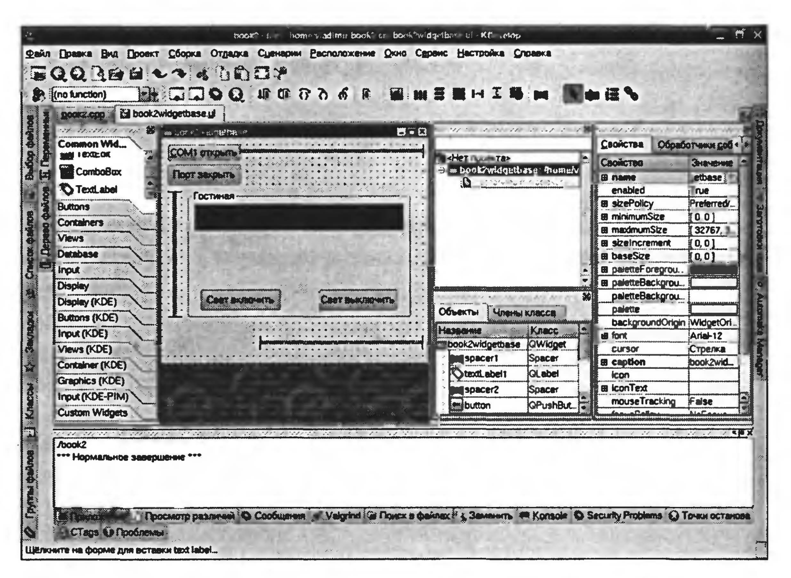


Рис. П.15. Изображение включенного света в программе

```
\class Posix_QextSerialPort
\version 0.70 (pre-alpha)
\author Wayne Roth
```

предлагает исходные тексты классов объектов для кроссплатформенного применения при работе с последовательным портом.

С объектами разных классов мы, в сущности, уже работали, когда создавали интерфейс. Почитав у Липпмана то, что относится к классовой сущности языка C++, я в какой-то мере осознал, что класс – это классно. Но как применить наверняка очень полезное и грамотное творение Вэйна Рота к моей простенькой программе, я понять не могу. Однако почти уверен, что есть разные пути и способы, и почти уверен, что я их не знаю. Для начала я просто создаю собственный класс, который добавляю в программу. Поскольку этот механизм работает, я, не мудрствуя лукаво, добавляю в программу исходные тексты программы, написанные Вэйном Ротом. Для моих целей хватает `posix_qextserialport` и `qextserialbase`. Я добавляю в проект и файлы заголовков, и основные файлы.

Второй, быть может не менее важный для меня момент осознания, – куда и что добавлять? Я понимаю, что следует записать:

```
#include <posix_qextserialport.h>
#include <qextserialbase.h>
```

но куда? И опять я, не мудрствуя лукаво, добавляю эти записи в файл, где происходят все события моей программы, – bookwidget.cpp и его файл заголовка. Не знаю, насколько это правильно, но это срабатывает. Программа транслируется без ошибок, проект собирается. Кстати, проект, который я собирал, я назвал так же, как в Visual Basic – barby (по имени богатой куклы – Barbie).

К этому проекту и обратимся в дальнейшем. Его графический интерфейс выглядит практически так же, как выше приведенный. Файл barbywidget.cpp (аналогично файлу bookwidget.cpp) выглядит следующим образом.

Вторая версия основной программы на языке C++

```
/***************************************************************
* Copyright (C) 2006 by Vladimir Gololobov                     *
* vgololobov@yandex.ru                               *
*                                      *
* This program is free software; you can redistribute it
and/or modify *
* it under the terms of the GNU General Public License
as published by  *
* the Free Software Foundation; either version 2 of the
License, or *
* (at your option) any later version.                     *
*                                      *
* This program is distributed in the hope that it will
be useful,    *
* but WITHOUT ANY WARRANTY; without even the implied
warranty of    *
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the     *
* GNU General Public License for more details.             *
*                                      *
* You should have received a copy of the GNU General
Public License*
```

```
* along with this program; if not, write to the          *
* Free Software Foundation, Inc.,                      *
* 59 Temple Place - Suite 330, Boston, MA  02111-1307,
USA.     *
*******************************************************/

#include <qlabel.h>
#include <posix_qextserialport.h>
#include <qextserialbase.h>
#include "barbywidget.h"

char command_on[7] = "R14$1N";
char command_of[7] = "R14$1F";

Posix_QextSerialPort comPort("/dev/ttyS0");
// Это позволяет использовать готовый класс.

barbyWidget::barbyWidget(QWidget* parent, const char*
name, WFlags fl)
: barbyWidgetBase(parent,name,fl)
{
   light->setEnabled(FALSE);//Здесь light - этикетка с
желтой картинкой.
}

barbyWidget::~barbyWidget()
{}

/*$SPECIALIZATION$*/
void barbyWidget::port_open_clicked()
{
   comPort.setName("/dev/ttyS0"); // Далее следуют
установки порта COM1.
   comPort.setDataBits(DATA_8);
   comPort.setStopBits (STOP_1);
   comPort.setBaudRate (BAUD2400);
   comPort.open(0);

   if (comPort.isOpen())
   {
      label->setText( "port open" ); // Это переделанная
этикетка.
   }
```

```
}

void barbyWidget::port_close_clicked()
{
   comPort.close();
   if (!comPort.isOpen())
      label->setText( "port close" );
}

void barbyWidget::light_on_clicked()
{
   comPort.writeBlock(command_on, 6);
   light->setEnabled(TRUE);
}

void barbyWidget::light_off_clicked()
{
   comPort.writeBlock(command_of, 6);
   light->setEnabled(FALSE);
}
#include "barbywidget.moc"
```

Файл заголовка barbywidget.h:

```
/***************************************************************************
* Copyright (C) 2006 by Vladimir Gololobov                              *
* vgololobov@yandex.ru                                   *
*                                   *
* This program is free software; you can redistribute it
and/or modify *
* it under the terms of the GNU General Public License
as published by  *
* the Free Software Foundation; either version 2 of the
License, or *
* (at your option) any later version.                  *
*                                   *
* This program is distributed in the hope that it will
be useful,    *
* but WITHOUT ANY WARRANTY; without even the implied
warranty of    *
* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
See the    *
* GNU General Public License for more details.         *
*                                   *
```

```
* You should have received a copy of the GNU General
Public License *
* along with this program; if not, write to the        *
* Free Software Foundation, Inc.,                      *
* 59 Temple Place - Suite 330, Boston, MA  02111-1307,
USA.     *
***********************************************************/
#ifndef _BARBYWIDGET_H_
#define _BARBYWIDGET_H_

#include "barbywidgetbase.h"
#include <posix_qextserialport.h>
#include <qextserialbase.h>

class barbyWidget : public barbyWidgetBase
{
Q_OBJECT

public:
barbyWidget(QWidget* parent = 0, const char* name = 0,
WFlags fl = 0 );
~barbyWidget();
/*$PUBLIC_FUNCTIONS$*/

public slots:
/*$PUBLIC_SLOTS$*/
virtual void port_open_clicked();  // Это бывшая клавиша
buttom.
virtual void port_close_clicked(); // Далее добавленные
клавиши.
virtual void light_on_clicked();
virtual void light_off_clicked();
protected:
/*$PROTECTED_FUNCTIONS$*/

protected slots:
/*$PROTECTED_SLOTS$*/

};

#endif
```

После запуска программы, щелчка кнопки **COM1 открыть** и кнопки **Свет включить** получаем вид программы, показанный на рис. П.16.



Рис. П.16. Вид работающей программы

Первая неожиданность – программа работает, но модуль не реагирует на команды. Ситуация повторяет начало работы с портом в части проверки конвертера. Первое, что удается проверить с помощью мультиметра, – наличие изменений сигналов на входе конвертера. Если поменять порт (то есть /dev/ttyS1), напряжение на сигнальном выводе останется неизменным. Это хорошо. Но плохо, что модуль не реагирует на команды. На всякий случай я возвращаюсь к программе на Visual Basic и проверяю работу модуля. Сомнений не остается – модуль отрабатывает команды.

Пользуясь тем, что модуль работает, проверяю сигналы интерфейса RS485 в программе, работающей с Visual Basic. Затем я перезагружаю операционную систему и запускаю

программу из KDevelop. Спустя некоторое время, удается заметить, что сигнал RTS на 3 выводе микросхемы MAX1483 сбрасывается в «0» при открытии порта. Добавление в программу в разделе настроек после открытия порта исправляет ситуацию. Модуль выполняет команды.

```
comPort.open(0);
comPort.setRts (FALSE); // Добавленная строка
```

Теперь остается разобраться с командами запроса статуса реле. Модифицированная программа выглядит следующим образом:

```
#include <qlabel.h>
#include <posix_qextserialport.h>
#include <qextserialbase.h>
#include "barbywidget.h"

char command_on[7] = "R14$1N";
char command_of[7] = "R14$1F";
char command_st[7] = "R14$1S";
char answer_st[19] = "
int i = 0;

Posix_QextSerialPort comPort("/dev/ttyS0");

barbyWidget::barbyWidget(QWidget* parent, const char*
name, WFlags fl)
: barbyWidgetBase(parent,name,fl)
{
light->setEnabled(FALSE);
}

barbyWidget::~barbyWidget()
{}

/*$SPECIALIZATION$*/
void barbyWidget::port_open_clicked()
{
comPort.setName("/dev/ttyS0");
comPort.setDataBits(DATA_8);
comPort.setStopBits (STOP_1);
```

```
comPort.setBaudRate (BAUD2400);
comPort.open(0);
comPort.setRts (FALSE);
answer->setText("              ");
if (comPort.isOpen())
{
label->setText( "port open" );
}
}

void barbyWidget::port_close_clicked()
{
comPort.close();
if (!comPort.isOpen())
label->setText( "port close" );
answer->setText("                     ");
}

void barbyWidget::light_on_clicked()
{
comPort.writeBlock(command_on, 6);
comPort.writeBlock(command_st, 6);
while (!comPort.atEnd());
for (i=0;i<100000000;++i);
comPort.readBlock(answer_st, 18);
answer->setText(answer_st);
if (answer_st[17] == "N")
light->setEnabled(TRUE);
}

void barbyWidget::light_off_clicked()
{
comPort.writeBlock(command_of, 6);
comPort.writeBlock(command_st, 6);
while (!comPort.atEnd());
for (i=0;i<100000000;++i);
comPort.readBlock(answer_st, 18);
answer->setText(answer_st);
if (answer_st[17] == "F")
light->setEnabled(FALSE);
}
#include "barbywidget.moc"
```

В основном, изменения касаются чтения ответа в буфер порта после отправки запроса:

```
while (!comPort.atEnd());
for (i=0;i<100000000;++i);
comPort.readBlock(answer_st, 18);
```

Пустой цикл for добавлен, чтобы ответ был полностью получен. Массив char answer_st[19] служит для чтения буфера ввода порта, и сохраняемые им команды содержат все, что было отправлено (рис. П.17).



Рис. П.17. Прием ответа модуля в программе

В правом нижнем углу формы отображается содержание этого массива. Поэтому в проверке его содержимого я проверяю только последний символ:

```
if (answer_st[17] == "N")
```

Возможно, не лучшим образом, но механизм работает. По команде выключения света свет на картинке выключается.

На макете, естественно, по команде включения света загорается индикатор, который гаснет по команде выключения света.

Запланированная часть работы выполнена.

Две полезные схемы

Первая схема относится к настенному выключателю, работающему по протоколу X10. Что полезного можно почерпнуть из этой схемы? Например, организацию сканирования сети и управления триаком. Схему я привожу, как она сохранилась в архиве (рис. П.18).

Вторая схема – датчик движения. Датчик работает по радиоканалу. Стоит он в Москве 36 долларов. Возможно, есть и более низкие цены на эти датчики. Если вам захочется поэкспериментировать с датчиками движения, и вы можете себе позволить купить его, он может оказаться более разумным решением, чем сборка собственного датчика движения, – линза, мне кажется, сложна для самостоятельного изготовления, и может стоить достаточно дорого. Кроме того, можно использовать в экспериментах передатчик, которым снабжено устройство. Протокол X10 использует разные команды для IR- и RF-посылок, если я не ошибаюсь. Но с этим можно разобраться на месте, если вы захотите использовать датчик движения вместе с его радиоканалом (рис. П.19).

Разветвитель видеосигнала

Подключение бытовой аудиоаппаратуры к системе, думаю, не вызовет затруднений. Даже к линейному выходу музыкального центра или видеомагнитофона можно параллельно включить несколько приемников (телевизоров или управляемых усилителей) без заметного ухудшения качества звука. Если же вы используете системный аудиокоммутатор, добавить развязывающие усилители тоже несложно. Можно использовать транзисторы, проверить работу предполагаемой развязки в программах, о которых я говорил выше, и добавить к схеме коммутатора дополнительные элементы.

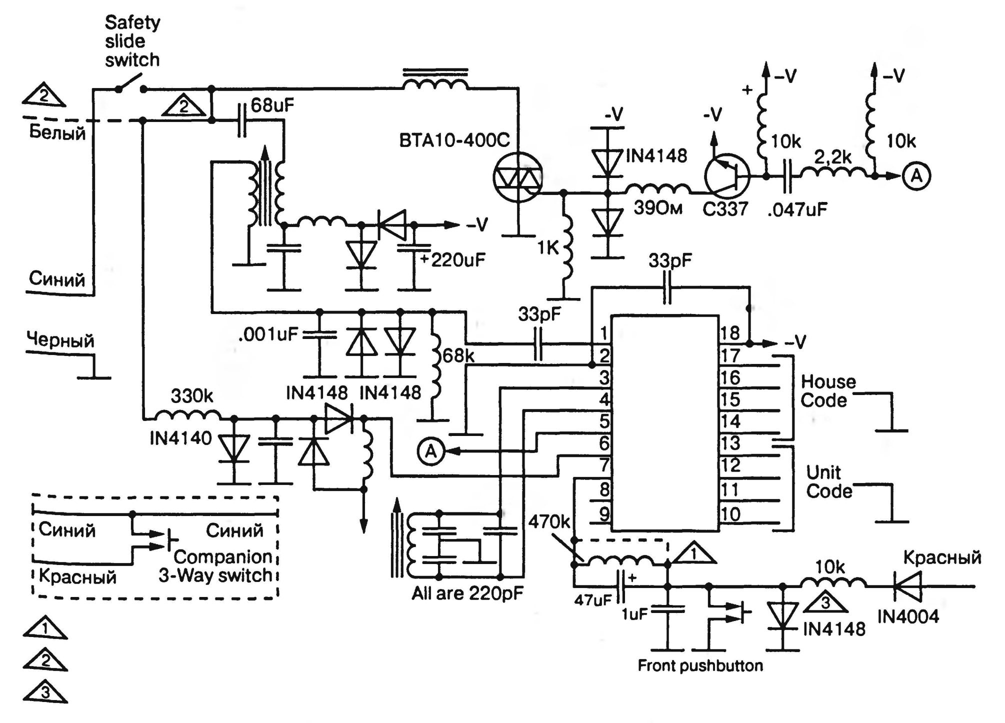


Рис. П.18. Схема диммера WS467

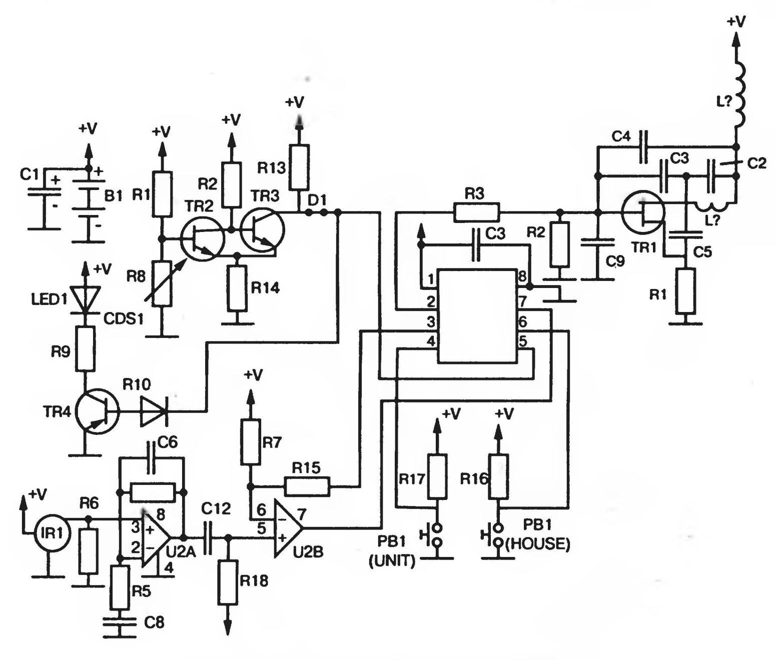


Рис. П.19. Схема датчика движения MS12A

Еще проще можно организовать работу, применяя операционные усилители. Практически любой из них обеспечит необходимую полосу пропускания. Усиления не требуется, а добавить регулировку усиления можно с целью обеспечения одинаковой громкости всех приемников сигнала.

Немного сложнее обстоят дела с «перемещением» видеосигнала. По этой причине я привожу схему, опубликованную в журнале «Радио» (рис. П.20).

Как правило, выходное сопротивление всех источников видеосигнала рассчитано на подключение коаксиального кабеля с волновым сопротивлением 75 Ом. Такое же входное сопротивление имеют приемники.

Если для передачи сигнала вы используете коаксиальный кабель, использование разветвителя ясно. Но, если у вас нет желания вместе с проводами системы укладывать еще и коаксиальный кабель, для системных нужд можно использовать

Рис. П.20. Схема разветвителя видеосигнала

витую пару в кабеле (CAT 5), имеющем 4 витые пары, а оставшиеся пары применить для передачи видеосигнала. Волновое сопротивление витой пары, если я не ошибаюсь, – 120 Ом. Разветвитель видеосигнала в этом случае должен иметь резисторы на выходе микросхем 120 Ом. Входное сопротивление приемников желательно тоже увеличить до 120 Ом, иначе, в этом я убедился на собственном опыте, второе изображение на экране телевизора будет слишком заметно. Оно возникает из-за отражений в несогласованной линии.

Особенно удобно использовать разветвитель подобной конструкции, когда в одно помещение вы проложили коасиальный кабель, а в другое вынуждены передавать сигнал по витой паре. В такой ситуации просто установите одно из сопротивлений на выходе микросхемы 75 Ом (в оригинале 56 Ом), а другое – 120 Ом. Или примените две микросхемы. Возможность менять усиление в активном разветвителе сигнала оказывается не только полезной, но и необходимой.

Есть недорогой набор, предлагаемый фирмой Мастер КИТ, – NM2901. С его помощью вы соберете разветвитель видеосигнала гораздо быстрее. Есть и другое решение – применить конвертер для преобразования видеосигнала в радиосигнал на частоте одного из свободных каналов телевидения,

а этот радиосигнал подмешать к эфирному телевидению. Теперь обычный «краб», который у вас уже работает, «переместит» видеосигнал по всему дому.

Конвертер сделать несколько сложнее, но его преимущество в том, что вы можете к радиосигналу добавить и звуковой сигнал. Отпадает необходимость в заботах о коммутации и передаче звука. Можно также купть готовые конвертеры подобного рода, но стоят они дороже, чем пара микросхем.

Схемы для экспериментов с радиоканалом

Если вам захочется провести эксперименты с радиоканалом вместо проводной связи модулей, то:

- лучше было бы воспользоваться готовыми радиомодулями, но дорого;
- не забывайте, что ваши эксперименты могут мешать вашим соседям;
- не забудьте о программах, о которых говорилось выше;
- проверяйте все решения на устойчивость;
- может существовать множество интересных решений, что само по себе – целое «поле чудес» для творчества.

Схемы, приведенные на рис. П.21, надеюсь, помогут вам начать эксперименты. Я не помню источник, откуда появились эти схемы в моем архиве, думаю из литературы по радиоуправляемым моделям. Хорошие решения можно найти в соответствующей литературе. Интересно было бы использовать цифровые схемы для этих целей. Я встречал подобные устройства в Интернете, но эксперименты в этой области не проводил, поэтому не готов предложить готовое решение.

Элементы схемы: C1 = 0,1 мкФ, C2–C4 = 200 пФ, C5 = 1000 пФ, C6–C7 = 5–20 пФ, C8 = 10 пФ.

R1 = 2,4 кОм, R2 = 4,3 кОм, R3 = 2,4 кОм, R4 = 4,7 кОм, R5 = 100 Ом.

L1, L2, L5 – дроссели индуктивностью 50 мкГн (50–60 витков провода ПЭЛШО 0,1, намотанных виток к витку на каркасе диаметром 5 мм).

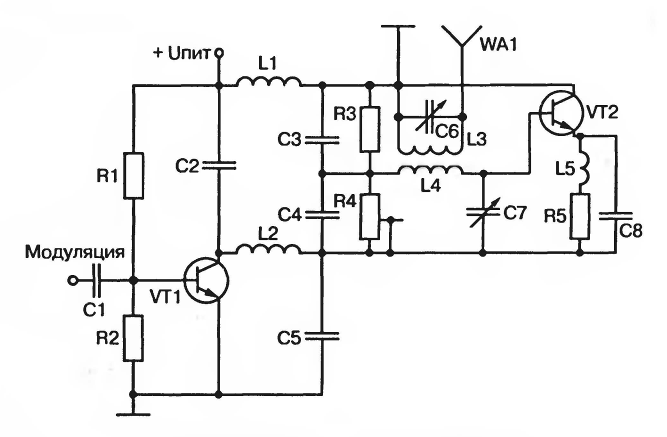


Рис. П.21. Схема АМ прередатчика на 27 МГц

L3–L4 – катушки, намотанные на одном каркасе диаметром 10мм проводом ПЭЛШО 0,35 виток к витку и имеющие 6 и 8 витков.

VT1 – транзистор практически любого типа, например КТ3102, VT2, высокочастотный транзистор типа ГТ311 или аналогичный.

Напряжение питания Uпит = 4–5 В.

Режим питания генератора высокой частоты, собранного на транзисторе VT2, определяется значениями резисторов делителя R1R2, а режим работы самого транзистора VT2 – значениями резисторов делителя R3R4 и резистора R5, который задает, в основном, рабочий ток транзистора VT2.

Конденсаторами C6 и C7 передатчик настраивается на частоту 27 МГц.

На работу высокочастотного генератора большое влияние оказывает выбор транзистора VT2, который должен быть достаточно высокочастотным, в противном случае генератор может не возбуждаться на частоте 27 МГц.

Назначение генератора, собранного на транзисторе VT2, – выработать сигнал высокой частоты, который станет «несущей» для информационного сигнала, подаваемого на вход, обозначенный на схеме как «модуляция». Именно модулирующий сигнал и передаст информацию, которую мы хотели бы перенести «через эфир»: речь диктора или музыку

вещательной радиостанции, сигнал дистанционного управления моделью и т.п.

Модуляция высокочастотного сигнала информационным происходит благодаря изменению амплитуды несущего сигнала при изменении питающего напряжения генератора. Транзистор VT1, на вход которого поступает медленно изменяющийся (по сравнению с генерируемым сигналом) информационный сигнал, в такт с ним изменяет напряжение питания задающего генератора, тем самым «модулируя», наполняя нужной нам информацией сигнал с частотой 27 МГц.

Модулированный высокочастотный сигнал через антенну WA1 излучается в пространство и может быть принят приемником, настроенным на частоту, на которой работает высокочастотный генератор.

Конечно, для устойчивой работы передатчика на выбранной частоте желательно стабилизировать высокочастотный генератор кварцевым резонатором, но для экспериментальных целей это не обязательно.

Если у вас нет опыта по работе с радиоустройствами, я бы посоветовал следующий порядок работы: использовать программы, о которых я рассказывал, для проверки работы схемы на компьютере. Изменяя величину резистора R4 в делителе R3R4, постараемся определить, что будет происходить с током в эмиттерной цепи транзистора VT2 (рис. П.22).

Схему я несколько изменил, приблизив ее к решению, которое предполагается использовать в экспериментах. В первом случае резистор имеет одно из крайних значений. Ток, который показывает амперметр, составляет доли микроампера. Переведем его в другое крайнее значение (рис. П.23).

Ток во втором положении резистора при наличии ВЧ-сигнала значительно возрастает. Теперь, собрав передатчик, постараемся настроить его так, чтобы ток в измеряемой цепи составил 10–20 мА. Есть надежда, что при этом передатчик излучает сигнал.

Полезно воспользоваться простой схемой для проверки работы передатчика (рис. П.24).

Компоненты схемы – WA1, тонкий металлический штырь длиной 20–30 см; C1 3пФ, C2 конденсатор переменной емкости с воздушным диэлектриком емкостью 5–20 пФ, C3 150 пФ;

Рис. П.22. Ток эмиттера при отстутствии ВЧ-сигнала

Рис. П.23. Ток эмиттера при наличии ВЧ-сигнала

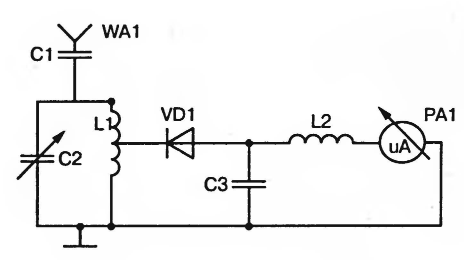


Рис. П.24. Индикатор поля

VD1 Д2В; катушка L1, намотанная на каркасе 12 мм и имеющая 12–14 витков провода ПЭВ–1 0.8 с отводом от середины, индуктивность катушки L2 30 мкГн, PA1 микроамперметр на ток полного отклонения 100 мкА. В качестве микроамперметра можно использовать мультиметр, включенный в режим измерения тока.

По сути, индикатор поля – детекторный приемник, нагруженный на микроамперметр. Можно вывести ручку конденсатора входного контура (C2) на переднюю панель и проградуировать индикатор с помощью генератора стандартных сигналов, к выходу которого подключают отрезок монтажного провода длиной 20–30 см, а антенну индикатора располагают рядом. Изменяя частоту генератора, настраивают индикатор на максимальное отклонение стрелки микроамперметра и считывают частоту со шкалы генератора.

Для экспериментов можно использовать входную часть приемника.

Соберем приемник по схеме, показанной на рис. П.25.

Схема приемника (данные можно использовать те же, что и у передатчика на 27 МГц, а конденсатор C1 взять емкостью в 100 пФ) предлагается для экспериментов. Может быть использована любая другая. В качестве транзисторов VT1, VT2 можно задействовать транзисторы ГТ311 или аналогичные. Для соединения радиоприемника с микроконтроллером PIC16F628A интересно было бы использовать встроенный компаратор или установить внешний. В крайнем случае, использовать быстродействующий операционный усилитель в качестве компаратора или микросхемы серии К561. Здесь много вариантов, главное добиться стабильной работы радиоканала.

WA1 C3 C1 R1 C2 VT1 C4 R2 R4 C6 C8 + +Uпит VD1 VD2 VT2 C9 C5 R5 C7 R6 Выход

Рис. П.25. Схема приемника на 27 МГц для экспериментов с радиоканалом

После сборки приемника, используя в качестве модулирующего сигнала передатчика генератор на 400 Гц, следует подключить приемник к усилителю и настроить его так, чтобы тональный сигнал был слышен при разнесении приемника и передатчика на расстояние в несколько метров. Хорошо бы настроить передатчик на 27 МГц, но как это сделать без измерительных приборов, я не знаю. Лучше не увлекаться мощность передатчика, ограничив расстояние уверенного приема несколькими метрами.

Убедившись в работоспособности радиоканала, можно приступить к экспериментам по передаче управляющих сигналов. Не забудьте, что при отсутствии управляющих сигналов, передатчик должен быть выключен.

Немного о программировании на C++

Поскольку при программировании микроконтроллера я использовал язык С, мне показалось уместным добавить хотя бы несколько слов о языке. Но я не сделаю это лучше, чем С. Липпман. Когда мне понадобилось познакомиться с языком, я использовал оригинальную версию его книги «Essential C++». Фрагменты моего конспекта (или перевода) я включаю в Приложение, но советую эту книгу приобрести.

Как писать программы на C++

Положим, нам нужно написать простую программу, отправляющую сообщение на терминал пользователя, которое просит ввести имя. Мы прочитываем введенное имя, сохраняем, чтобы использовать в дальнейшем, и, наконец, приветствуем пользователя по имени.

Прекрасно, откуда начнем? Начнем там, откуда начинаются все программы на C++ – с функции, называемой `main()`. `main()`. Внедряемая пользователем функция в следующей основной форме:

```
int main(){
// код нашей программы далее}
```

где `int` – это ключевое слово языка C++. *Ключевые слова* – предопределенные имена, имеющие особое значение в языке. `int` представляет встроенный целый тип данных.

Функция – независимая последовательность кодов, производящих некие выкладки. Она состоит из частей: возвращаемого типа, имени функции, списка параметров и тела функции. Рассмотрим каждую из нихх по очереди.

Возвращаемый тип функции обычно представляет результат вычислений (выкладок). `main()` имеет целый возвращаемый тип. Значение, возвращаемое `main()`, показывает, было ли выполнение нашей программы успешным. По соглашению `main()` возвращает «0» при удачном выполнении. Ненулевое значение показывает, что что-то идет не так.

Имя функции выбирается программистом из соображений наилучшим образом определить, что функция будет делать. `min()` и `sort()`, например, – хорошие имена функций. `f()` и `g()` не так хороши. Почему? Они менее информативны в отношении назначения функции.

`main` – не ключевое слово. Система компиляции, выполняющая нашу программу C++, предполагает, что функция `main()` предопределена. Если мы забудем включить ее, программа не будет работать.

Список параметров функции заключен в круглые скобки и размещен за именем функции. Пустой список параметров,

как у `main()`, обозначает, что параметры функции не передаются.

Список параметров, – обычно разделяемый запятой перечень типов данных, которые пользователь может передать функции при ее выполнении. (Мы говорим, что пользователь вызывает или активизирует функцию). Например, если мы напишем функцию `min()`, возвращающую наименьшее из двух значений, ее список параметров должен определить типы двух значений, которые мы сравниваем. Функция `min()` для сравнения двух целых чисел может быть определена следующим образом:

```
int min(int val1, int val2){
// код программы следует здесь ...}
```

Тело программы заключено в фигурные скобки `{}`. Оно содержит последовательность кодов, которая осуществляет все действия функции. Двойная прямая косая черта `//` означает пояснения, заметки программиста по каким-то аспектам кодов. Они предназначены для читающих исходный текст программы и выбрасываются при компиляции. Все, следующее за прямой двойной косой чертой до конца линии, относится к комментарию.

Наша первая задача – написать вывод сообщения на терминал пользователя. Ввод и вывод в языке C++ не предопределены. Вернее, они поддерживаются объектно-ориентированным классом иерархии, встроенным в C++ как часть стандартной библиотеки языка.

Класс – тип данных, определяемых пользователем. Механизм работы классов – метод добавления типа данных, распознаваемого нашей программой. Объектно-ориентированная иерархия классов определяет семейство относительных типов класса, например терминала и файлового ввода, терминала и файлового вывода и т.д.

C++ предопределяет небольшое множество основных типов данных: булевы, символьные, целые и с плавающей точкой. Хотя они служат основой для всего программирования, мы не будем сосредотачивать на них внимание. Кинокамера, например, должна иметь положение в пространстве, обычно представляемое тремя числами в формате с плавающей

точкой. Камера должна, кроме того, иметь ориентацию в пространстве, описываемую еще тремя числами с плавающей точкой. Есть еще одно отношение, описывающее отношение расстояния до камеры к высоте предмета. Оно представлено единственным числом с плавающей точкой.

На очень примитивном уровне можно сказать, что камера представлена семью числами с плавающей точкой, шесть из которых образуют две группы координат x, y, z. Программирование на этом низком уровне требует, чтобы мы направляли нашу мысль то назад, то вперед – от манипуляций с абстрактной камерой к манипуляции с семью числами с плавающей точкой, представляющими камеру в программе.

Механизм классов позволяет нам добавить в программу уровни абстракции типов. Например, мы можем определить класс трехмерных точек для представления локализации и ориентации в пространстве или класс `Камер`, состоящий из двух объектов класса трехмерных точек и числа с плавающей точкой. Мы все еще представляем камеру семью числами с плавающей точкой. Но разница в том, что в нашей программе мы теперь можем напрямую манипулировать с классом `Камер`, а не с семью числами с плавающей точкой.

Определение класса обычно разбивается на две части, каждая из которых представлена своим файлом: файлом заголовка, производящим объявление операций, поддерживаемых классом, и файлом текста программы, который содержит реализацию этих операций.

Для использования класса мы включаем в программу файл заголовка. Файл заголовка делает класс узнаваемым программой. Стандартные библиотеки ввода/вывода в C++ вызываются из библиотеки `iostream`. Она содержит подборку связанных классов, поддерживающих ввод и вывод на пользовательский терминал и в файлы. Чтобы использовать библиотеку класса `iostream`, мы должны включить ассоциированный с ней файл заголовка:

```
#include <iostream>
```

Для обращения к терминалу пользователя мы используем предопределенный объект класса, названный `cout` (произносится «си аут»). Мы направляем данные, которые хотим,

чтобы cout написал, пользуясь оператором вывода (<<), следующим образом:

```
cout << "Please enter your first name: ";
```

Это представлено выражением программы C++, маленьким независимым кусочком программы на C++. Это аналог предложения в обычном языке. Выражение завершается точкой с запятой. Наше выражение пишет строчку символов (отмеченных кавычками) на терминале пользователя. Кавычки идентифицируют строку, они не отображаются на терминале. Пользователь видит:

```
Please enter your first name:
```

Наша следующая задача – прочитать ввод пользователя. Прежде чем мы сможем прочитать имя, напечатанное пользователем, мы должны определить объект, в котором сохраним информацию. Мы определим объект, обозначив тип данных объекта и дав ему имя. Мы уже видели один тип данных int. Но трудновато использовать его для хранения чьего-либо имени. Больше подходит тип данных из стандартной библиотеки строкового класса:

```
string user_name;
```

Это определяет user_name, как объект строкового класса. Определение названо заявительным предложением. Это предложение не будет приниматься до тех пор, пока мы не сделаем строковый класс известным программе. Мы осуществляем это включением файла заголовка строкового класса:

```
#include <string>
```

Для чтения ввода с терминала пользователя, мы используем предопределенный объект класса, называемый cin (произносится «си ин»). Мы используем оператор ввода (>>) для направления cin на чтение данных с терминала пользователя в объект подходящего типа:

```
cin >> user_name;
```

Последовательности вывода и ввода появятся на терминале пользователя следующим образом (ввод пользователя отмечен жирным шрифтом):

```
Please enter your first name: anna
```

Все, что нам остается сделать, – поприветствовать пользователя по имени. Мы хотим, чтобы вывод выглядел так:

```
Hello, anna ... and goodbye!
```

Понимаю, это не верх совершенства. Но терпение – мы только начали. Позже мы научимся более изящным вещам.

Первый шаг для вывода приветствия – перенаправить его на следующую строку. Мы сделаем это, написав символ новой строки в `cout`:

```
cout << '\n';
```

Символы обозначаются двумя апострофами. Существует две разновидности символов: печатные символы: буквы (`'a'`, `'A'` и т.д.), числа и знаки препинания (`';'`, `'-'` и т.д.), и непечатные символы: символ перевода строки (`'\n'`) или табуляция (`'\t'`). Поскольку нет символьного представления непечатных символов, самый общий пример – символ перевода строки и табуляция, они представляются специальной последовательностью из двух символов.

Теперь, перейдя на новую строку, мы хотим воспроизвести Hello:

```
cout << "Hello, ";
```

Дальше нам нужно вывести имя пользователя. Оно сохранено в строковом объекте `user_name`. Как мы сделаем это? Так же, как и с другими типами:

```
cout << user_name;
```

Наконец, мы завершаем приветствие, сказав «прощай» (обратите внимание, что строка символов может содержать оба вида: как печатные, так и непечатные символы):

```
cout << " ... and goodbye!\n";
```

В основном, все встроенные типы выводятся тем же путем – помещаем значение справа от оператора вывода. Например,

```
cout << "3 + 4 = ";
cout << 3 + 4;

cout << "\n";
```

дает следующий вывод:

```
3 + 4 = 7
```

Так же, как мы определили новые типы класса для использования в приложении, проводим привязку оператора вывода для каждого класса. Это позволяет пользователям класса выводить его отдельные объекты совершенно одинаковым образом со встроенными типами данных.

Вместо написания отдельных выводов каждый на своей строке мы можем объединить их в одно предложение:

```
cout << "\n"
<< "Hello, "
<< user_name
<< " ... and goodbye!\n";
```

В завершении мы можем завершить `main()`, используя выражение `return` (возврат):

```
return 0;
```

`return` – ключевое слово C++. Выражение, следующее за ним, в данном случае «0», представляет результат выполнения функции. Напомню, что возвращаемое функцией значение «0» говорит об успешном завершении работы.

Соберем кусочки вместе – получим нашу первую завершенную программу на C++:

```
#include <iostream>
#include <string>
using namespace std; // пока не объясняю этого ...

int main(){
string user_name;
cout << "Please enter your first name: ";
```

```
cin >> user_name;
cout << "\n"
   << "Hello, "
   << user_name
   << " ... and goodbye!\n";
return 0;}
```

После компиляции при выполнении этот код произведет следующее (мой ввод подсвечен жирным шрифтом):

```
Please enter your first name: anna
Hello, anna ... and goodbye!
```

Одно выражение я не пояснил:

```
using namespace std;
```

Посмотрим, смогу ли я объяснить это, не напугав вас. (Сделайте глубокий вдох!). using и namespace – ключевые слова C++. std – имя стандартной библиотеки namespace. Все, выбираемое из стандартной библиотеки (как объекты строкового класса, класса iostream – cout и cin) заключено внутри, std namespace. Конечно, ваш следующий вопрос: а что такое namespace?

namespace – метод упаковки библиотечных имен, при котором они могут быть использованы в окружении пользовательской программы без появления коллизий. (Коллизия имен обнаруживается, когда есть два использования одного и того же имени в приложении, так что программа не может различить их. Когда такое случается, программа не может продолжать работу, пока не разрешится конфликт имен). Namespaces – способ, ограждающий видимость имен.

Для использования объектов строкового класса и класса iostream – cin и cout – внутри нашей программы мы должны не только включить файлы заголовков string и iostream, но так же сделать имена видимыми с std namespace. Использование директивы:

```
using namespace std;
```

простейший метод сделать имена внутри namespace (пространство имен) видимыми.

Определение и инициализация объектов данных

Теперь, чтобы завладеть вниманием пользователя, выполним короткий тест. Мы отобразим два числа из числовой последовательности и предложим пользователю угадать следующие значения в последовательности. Например,

```
The values 2,3 from two consecutive elements of a
numerical sequence.
What is the next value?
```

Эти значения – третий и четвертый элементы из последовательности Фибоначчи: 1, 1, 2, 3, 5, 8, 13 и т.д. Последовательность Фибоначчи начинается с двух элементов – единиц. Каждый следующий элемент это сумма двух предшествующих.

Если пользователь введет 5, мы поздравим его и спросим, хочет ли он попробовать другую числовую последовательность. Любое другое введенное значение – неверно, и мы спросим пользователя, хочет ли он погадать еще.

Чтобы поддержать интерес к программе, мы сохраним текущий счет, основанный на отношении правильных ответов к числу попыток.

Программа нуждается, по меньшей мере, в пяти объектах: объекте строкового класса для сохранения имени пользователя, трех целых объектах класса для запоминания по очереди попыток, числа попыток, числа успешных попыток, и объект класса чисел с плавающей точкой для запоминания счета.

Для определения объектов данных мы должны ввести имена и тип данных. Имена могут быть любыми комбинациями букв, цифр, подчеркиваний. Буквы регистрозависимые. Каждое из имен `user_name`, `User_name`, `uSeR_nAmE` и `user_Name` относится к разным объектам.

Имя не должно начинаться с цифры. Например, `1_name` неправильно, `name_1` – правильно. Также имя не должно совпадать с ключевыми словами языка. Например, `delete` – ключевое слово языка, так что мы не должны использовать

его в нашей программе. (Это объясняет, почему оператор удаления символа из строкового класса – это erase(), а не delete()).

Каждый объект должен быть своего типа данных. Имя объекта позволяет нам обратиться к нему непосредственно. Тип данных определяет область значений, сохраняемых объектом, и количество памяти, необходимой для запоминания этого значения.

Мы видели определение user_name в предыдущем разделе. Перенесем то же определение в новую программу:

```
#include <string>
string user_name;
```

Класс – программно-определенный тип данных. С++ также поддерживает множество встроенных типов данных: булевы, целые, с плавающей точкой и символьные. Ключевые слова, ассоциированные с каждым из них, позволяют нам определить тип данных. Например, для запоминания значения, введенного пользователем, мы определим объект целого типа:

```
int usr_val;
```

int – ключевое слово языка определяющее, что объект usr_val – целого типа. Оба объекта: число попыток, сделанных пользователем и число правильных ответов, – объекты целого типа. Разница только в том, что мы бы хотели дать им начальное значение «0». Мы можем вывести каждое на отдельную строку:

```
int num_tries = 0;
int num_right = 0;
```

или определить их в одной строке через запятую:

```
int num_tries = 0, num_right = 0;
```

В общем, лучше придерживаться правила инициализировать объект данных, даже если значение только обозначает,

что объект не имеет полезного значения вовсе. Я не инициализировал usr_val, потому что значение будет получено непосредственно из пользовательского ввода прежде, чем программа как-то использует объект.

Альтернативный вариант инициализации – использовать так называемый конструкционный синтаксис

```
int num_tries(0);
```

Почему есть два инициализационных синтаксиса? Хуже того, почему я сейчас об этом говорю? Что ж, посмотрим, отвечает ли мое объяснение на оба вопроса.

Использование оператора присваивания для инициализации пришло из языка С. Оно хорошо работает с объектами данных встроенных типов и классами объектов, которые могут быть инициализированы единственным значением, например строковым классом:

```
string sequence_name = "Fibonacci";
```

Однако данный способ не так хорош для класса объектов, которые требуют нескольких значений для инициализации, как, например, класс стандартной библиотеки комплексных чисел, где каждое требует двух значений: первое – для действительной части, второе – для мнимой. Альтернативный конструкционный синтаксис был введен для поддержки многозначной инициализации:

```
#include <complex>
complex<double> purei(0, 7);
```

Странная нотация скобок, следующая за complex, означает, что класс комплексных чисел – класс шаблонов. Класс шаблонов позволяет нам определять класс без спецификации типа данных одного или всех членов класса.

Класс комплексных чисел, например, состоит из двух членов объекта данных. Один представляет реальную часть числа, второй – мнимую. Эти члены должны быть числами типа данных с плавающей точкой, но какого? C++ поддерживает три типа чисел с плавающей точкой: единичной точности, представляемых ключевым словом float; двойной точности, представленной ключевым словом double; и расширенной

точности, представленной двумя ключевыми словами long double.

Механизм класса шаблонов позволяет программисту откладывать определение типа данных, используя класс шаблонов. Это дает ему возможность вставить «заглушку», которую позднее он заполнит реальным типом данных. В предыдущем примере использовался выбор данных типа класса комплексных чисел double.

Что ж, возможно, появляется больше вопросов, чем получается ответов. Это происходит от того, что шаблоны, поддерживаемые С++, двух инициализационных синтаксисов для встроенных типов данных. Когда встроенные типы данных и программно определенный класс типов имеют разный инициализационный синтаксис, невозможно написать шаблон, который поддерживает и встроенный класс, и класс типа данных. Унификация синтаксиса упрощает разработку шаблонов. К сожалению, раскрытие синтаксиса приводит к появлению еще большей путаницы!

Счет пользователя должен быть значением с плавающей точкой, поскольку возникает нецелое отношение. Мы определим его типом double:

```
double usr_score = 0.0;
```

Нам также нужно сохранить место для ответов пользователя yes/no: Make another try? Try another sequence?

Мы можем сохранить ответы пользователя в символьном объекте данных:

```
char usr_more;

cout << "Try another sequence? Y/N? ";
cin >> usr_more;
```

Ключевое слово char относится к символьному типу. Символ окаймляется апострофами, обозначая 'a', '7', ';' и т.д. Некоторые специальные встроенные символьные обозначения приведены ниже (их иногда называют эскейп последовательности):

```
\n' newline (новая строка) "\t" tab (табуляция) "\0"
null (нуль) "\"' single quote (апостроф) "\"' double
quote (кавычки) "\\" backslash (обратная косая черта)
```

Например, для создания перевода на новую строку и табуляции перед вводом имени пользователя мы можем написать:

```
cout << "\n" << "\t" << user_name;
```

Можно также объединить в строку отдельные символы:

```
cout << "\n\t" << user_name;
```

Обычно мы используем эти специальные символы в буквенной строке. Например, для представления пути к файлу в системе Windows необходимо ставить обратные косые черточки:

```
"F:\\essential\\programs\\chapter1\\ch1_main.cpp";
```

C++ поддерживает встроенный булев тип данных для представления значений `true/false`. В нашей программе, например, мы можем определить булев объект для контроля над необходимостью вывода на дисплей следующей числовой последовательности:

```
bool go_for_it = true;
```

Булев объект обозначен ключевым словом `bool`. Он может сохранять два буквенных значения: `true` или `false`.

Все определенные объекты данных затем модифицируются в ходе нашей программы. `go_for_it`, например, в конечном счете, установится в `false`, а `usr_score` обновляется с каждой попыткой пользователя.

Иногда необходимо наличие объекта для представления постоянных значений: максимального числа попыток, определяемых для пользователя или значения числа «пи». Объекты, сохраняющие эти значения, не будут меняться по ходу программы. Как мы можем предотвратить случайное изменение данных объектов? Заручиться поддержкой языка, объявив эти объекты, как `const`:

```
const int max_tries =3;
const double pi = 3.14159;
```

Объекты `const` не могут изменяться после инициализации их значения. Любые попытки переустановить значение объекта `const` приведут к ошибке при компиляции. Например:

```
max_tries = 42; // error: объект const
```

Написание выражений

Встроенные типы данных поддерживаются набором операторов: арифметических, логических, отношения, структурообразующих. Арифметические операторы не имеют особенностей, исключая деление целых и получение остатка.

```
// Арифметические операторы
+ сложение a + b
- вычитание a - b
* умножение a * b
/ деление a / b
% остаток a % b
```

Деление двух целых значений выдает целое. Любой остаток отсекается, и это не округление. Остаток получается с помощью оператора %:

```
5 / 3 результат 1 тогда, как 5 % 3 имеет результат 2
5 / 4 результат 1 тогда, как 5 % 4 имеет результат 1
5 / 5 результат 1 тогда, как 5 % 5 имеет результат 0
```

В каких случаях мы нуждаемся в операторе получения остатка? Представьте, что мы хотим печатать не более восьми слов в линии. Если количество слов меньше восьми, мы выводим пробелы после слов. Если строка состоит из восьми или более слов, мы выводим переход на новую строку (`newline`):

```
const int line_size = 8;
int cnt = 1;
// это выражение выполняется много раз со
// строками, представленными разными значениями,
// и cnt увеличивается на единицу при каждом проходе...
cout << a_string<< (cnt % line_size ? " " : "\n");
```

Выражение в скобках за оператором вывода, похоже, не имеет для вас значения, пока вы не освоите условный оператор (`? :`). Результат выражения – либо пробел, либо переход на новую строку – в зависимости от того, нулевое или ненулевое значение принимает оператор остатка. Посмотрим, какой смысл мы можем ему придать.

Выражение `cnt % line_size` превращается в ноль, когда `cnt` достигает значения `line_size`; иначе оно не равно нулю.

Что это означает для нас? Условный оператор имеет основную форму:

```
expr? execute_if_expr_is_true: execute_if_expr_is_false;
выражение?выполнять_если_выражение_истинно:
выполнять_если_выражение_ложно;
```

Если expr (выражение) принимает значение true, выражение, следующее за знаком вопроса, выполняется. Если expr принимает значение false, выполняется выражение, следующее за двоеточием. В нашем случае выражение становится либо пробелом, либо переводом на новую строку для оператора вывода.

Условное выражение обрабатывается как ложное (false), если его значение равно нулю. Любое ненулевое значение принимается как true. В примере, пока cnt не делится на восемь, а результат не нулевой, выполняется переход оператора по условию true, и печатается пробел.

Оператор структурного присваивания дает краткую нотацию при применении арифметической операции для задания объектов. Например, чтобы не писать:

```
cnt = cnt + 2;
```

программист C++ обычно пишет:

```
cnt += 2; // прибавить 2 к данному значению cnt
```

Структурное присваивание применимо к каждому арифметическому оператору: +=, -+, /=, и %=.

Когда к объекту прибавляется или вычитается 1, программист C++ использует операторы увеличения или уменьшения:

```
cnt++; // добавляет 1 к значению cnt (инкремент)
cnt--; // вычитает 1 из значения cnt (декремент)
```

Существует префиксная (до) и постфиксная (после) разновидность операторов увеличения и уменьшения. В префиксном написании объект увеличивается или уменьшается на 1 до того, как его значение используется:

```
int tries = 0;
cout << "Are you ready for try #"<< ++tries << "?\n";
```

tries увеличивается на 1 до вывода его значения. В постфиксном написании значение объекта вначале используется в выражении, а затем увеличивается (или уменьшается) на 1:

```
int tries = 1;
cout << "Are you ready for try #"<< tries++ << "?\n";
```

Теперь значение tries выводится до того, как увеличивается на 1. В обоих примерах значение 1 выводится.

Каждый из операторов сравнения принимает значение true или false. Их всего шесть:

```
== равно a == b
!= не равно a != b
< меньше чем a < b
> больше чем a > b
<= меньше чем или равно a <= b
>= больше чем или равно a >= b
```

Вот пример того, как мы можем использовать оператор равенства (эквивалентности) для проверки ответа пользователя:

```
bool usr_more = true;
char usr_rsp;
// спросим пользователя, хочет ли он продолжить
// прочитаем ответ в usr_rsp
if (usr_rsp == "N") usr_more = false;
```

Условное выражение if выполняет предложение, следующее за ним, если выражение в скобках принимает значение true. В данном примере usr_more устанавливается в false, если usr_rsp равно 'N'. Если usr_rsp не равно 'N', не делается ничего. «От противного», используя оператор неравенства, можно написать так:

```
if (usr_rsp != "Y")usr_more = false;
```

Проблема с проверкой usr_rsp только на значение 'N' в том, что пользователь может ввести ответ в нижнем регистре, то есть 'n'. Мы должны распознать оба значения. Одна из стратегий – добавить else:

```
if (usr_rsp == "N")
usr_more = false;
```

```
else if (usr_rsp == "n")
usr_more = false;
```

Если usr_rsp равно 'N', то usr_more устанавливается в false и больше ничего не делается. Если оно не равно 'N', выполняется else. Если usr_rsp равно 'n', то usr_more принимает значение false. Если же значение usr_rsp не равно ни тому, ни другому, значение usr_more не определяется.

Одна из распространенных ошибок начинающих программистов – использование оператора присваивания при проверке равенства, как показано ниже:

```
// раз, и присвоили usr_rsp символ "N"
// а теперь всегда условие принимает значение true
if (usr_rsp = "N")

// ...
```

Логический оператор ИЛИ (||) предлагает альтернативный путь проверки истинности во множественных выражениях:

```
if (usr_rsp == "N" || usr_rsp == "n")usr_more = false;
```

Логический оператор ИЛИ становится истинным (true), если любое из выражений истинно. Левое выражение проверяется первым. Если оно истинно, следующее выражение не проверяется. В нашем примере usr_rsp проверяется на равенство 'n', только если оно не равно 'N'.

Логический оператор И (&&) становится истинным, только если оба выражения истинны. Например,

```
if (password &&validate(password) &&
(acct = retrieve_acct_info(password)))
// процесс доступа ...
```

Верхнее выражение проверяется первым. Если оно ложно, оператор И принимает значение false, а остальное выражение не проверяется. В данном случае информация доступа принимается только при введении правильного пароля.

Логический оператор НЕ (!) принимает значение true, если выражение, которому он принадлежит, имеет значение false. Например, вместо записи:

```
if (usr_more == false)
cout << "Your score for this session is " << usr_score
<< " Bye!\n";
```

мы можем написать:

```
if (! usr_more) ...
```

Оператор предшествования

Есть одна «заморочка» в использовании встроенных операторов – при комбинации нескольких операторов в одном выражении порядок выполнения операций определяется предустановленным уровнем приоритетности для каждого. Например, результат выражения 5 + 2 * 10 всегда равен 25 и никогда 70, поскольку оператор умножения имеет больший приоритет, чем оператор сложения. В итоге 2 всегда умножается на 10 прежде, чем складывается с 5.

Мы можем переопределить приоритет, взяв в скобки операцию, с выполнения которой хотели бы начать. (5 + 2) * 10, например, принимает значение 70.

Для операторов, о которых я говорил, предопределенные уровни приоритетности написаны ниже. Оператор, который выше, имеет больший приоритет, чем тот, что ниже. Операторы, расположенные в одну линию, имеют порядок определения слева направо.

```
логическое NOT
арифметическое (*, /, %)
арифметическое(+, -)
отношение (<, >, <=, >=)
отношение (==, !=)
логическое AND
логическое OR
присваивание
```

Например, для определения четности ival мы можем написать:

```
! ival % 2 // не совсем хорошо
```

Наше выражение проверяет результат оператора остатка. Если ival четно, результат нулевой и логический оператор НЕ становится истинным. Иначе, если результат ненулевой, логический оператор НЕ принимает значение false. Во всяком случае, так нам хотелось бы.

К сожалению, результат выражения совершенно иной. Наше выражение всегда будет ложным, исключая значение ival, равное нулю!

Более высокий приоритет оператора логического отрицания приводит к тому, что он выполняется первым, действуя на ival. Если ival имеет ненулевое значение, результат – ложный, в противном случае – истинный. Полученное значение становится левой частью операции получения остатка. False становится «0» при использовании в арифметических операциях, а true принимает значение «1». В результате порядка выполнения операций выражение превращается в 0%2 для любых значений ival за исключением нуля.

Хотя мы не хотели получить данный результат, он не является ошибкой, по крайней мере, языковой ошибкой. Это лишь неправильное представление нашей задуманной программной логики. Компилятор об этом, конечно, не догадывается. Порядок выполнения – это одна из причин, затрудняющих программирование на С++. Для правильного выполнения выражения мы должны изменить порядок выполнения с помощью скобок:

```
! (ival % 2) // ok
```

Чтобы избежать этих проблем, необходимо поближе познакомиться с порядком следования операторов в С++. Я вам не помощник, в том смысле, что раздел не содержит ни полного перечня операторов, ни полного представления порядка следования операций – того, что я показал, должно хватить только для начала.

Написание условий и создание циклов

По определению, выражения выполняются по разу по мере прохождения программы, начиная с первого выражения `main()`. В предыдущих разделах мы кратко говорили о выражении `if`. Оно позволяет нам выполнять по условию одно выражение или их последовательность, основываясь на вычислении истинности условия. Дополнение `else` дает возможность проверить несколько условий. Циклические выражения позволяют выполнять одно выражение или их последовательность, основываясь на вычислении истинности. Следующий псевдокод программы использует два цикла (#1 и #2), один условный оператор `if` (#5), один условный оператор `if-else` (#3), и второй условный оператор, называемый `switch` (переключатель) (#4).

```
// Pseudo code: Основная логика программы - пока
пользователь хочет угадывать последовательности
{ #1
вывести на дисплей последовательность
пока угадано неверно, и
пользователь хочет угадать еще раз

{ #2
прочитать гипотезу
увеличить счетчик попыток
если угадано правильно

{ #3
увеличить счетчик угадываний
установить got_it в true

} else {
выразить сожаление по поводу неудачной попытки сделать
запрос на основании текущего номера попыток пользователя

// #4 спросить пользователя, хочет ли он попытаться еще
раз и
//прочитать ответ, если пользователь отвечает нет

// #5 установить go_for_it в false
}}}
```

Условные выражения

Условия выражения `if` должны быть записаны в круглых скобках. Если они истинны, выражение, непосредственно следующее за `if`, выполняется:

```
// #5
if (usr_rsp == "N" || usr_rsp == "n")  go_for_it =
false;
```

Если должно выполниться несколько выражений, они должны быть заключены в фигурные скобки, следующие за `if` (это называется блоком выражений):

```
//#3
if (usr_guess == next_elem)
{
// начало блока выражений
num_right++;
got_it = true;} // окончание блока выражений
Общая ошибка начинающих - забыть отметить блок:
// р-раз: пропущена отметка блока
// только num_cor++ относится к if,
// a got_it = true; выполняется вне условия
if (usr_guess == next_elem)
num_cor++;
got_it = true;
```

Выполнение `got_it` отражает намерения программиста. К сожалению, оно не отражает поведение программы. Наращивание `num_cor` относится к условному оператору `if` и выполняется только тогда, когда попытка пользователя эквивалентна значению `next_elem`. `got_it`, однако, не относится к условному оператору, поскольку мы забыли обозначить два выражения как блок. В этом примере `got_it` всегда устанавливается в значение `true` независимо от того, что делает пользователь.

Условный оператор `if` так же поддерживает расширение `else`. `Else` представляет одно выражение или блок выражений, которые будут выполняться, если условие принимает значение `false`. Например,

```
if (usr_guess == next_elem)
{
```

```
// пользователь угадал
}
else {
// пользователь не угадал
}
```

Другое использование оператора else – объединить два условных выражения. Например, если попытка пользователя неудачна, мы хотим варьировать ответ в зависимости от числа попыток. Мы должны написать три проверки как независимые условные выражения:

```
if (num_tries == 1)cout << "У-ух! Хорошая попытка, но не
совсем.\n";
if (num_tries == 2)cout << "М-да. Извини. Опять
неправильно.\n";
if (num_tries == 3)cout << "А-а, это труднее, чем могло
показаться, не так ли?\n";
```

Однако только одно из трех условий может быть истинным одновременно. Если одно из условий верно, остальные должны быть ложны. Мы можем отобразить зависимость между выражениями if, объединив их вместе с помощью выражений if-else:

```
if (num_tries == 1) cout << "У-ух! Хорошая попытка, но
не совсем\n";
else
   if (num_tries == 2) cout << "М-да. Извини. Опять
неправильно.\n";
else
if (num_tries == 3)cout << "А-а, это труднее, чем могло
показаться, не так ли?\n";

else cout << "Похоже, дальше бесполезно!\n";
```

Первое условное выражение выполняется. Если оно истинно, выражение, следующее за ним, выполняется, а последовательность else-if – нет. Если же первое условное выражение ложно, выполняется следующее, затем следующее, пока одно из условий не становится истинным или num_tries не становится больше 3, – все условия ложны, и последнее else выполняется.

Один из неприятных аспектов использования вложенных if-else – трудности с правильной их логической организацией. Например, мы хотели бы использовать if-else для разделения логики программы на два случая: когда пользователь угадывает и когда он не угадывает. Первая попытка не работает совсем, как мы и планировали:

```
if (usr_guess == next_elem)
{
// пользователь угадывает
}
else if (num_tries == 1)
else // ... выводим вопрос
if (num_tries == 2)
else // ... выводим вопрос
if (num_tries == 3)
else // ... выводим вопрос
// ... выводим вопрос
// теперь спросим у пользователя, хочет ли он еще раз
угадать
// но только, если он не угадал
// р-раз! и куда же мы воткнем это?
```

Каждая пара else-if непреднамеренно была сделана альтернативой удачной попытки. В результате нам некуда поместить вторую часть кода для поддержки неудачных попыток. Вот правильная организация:

```
if (usr_guess == next_elem){
// пользователь угадал} else { // пользователь не угадал
if (num_tries == 1)
else // ...
 if (num_tries == 2)
else // ...
if (num_tries == 3)
// ...
else // ...

cout << "Желаете попробовать еще? (Y/N) ";
char usr_rsp;
cin >> usr_rsp;
if (usr_rsp == "N" || usr_rsp == "n") go_for_it =
false;}
```

Если значение проверяемого условия имеет интегральный тип, мы можем заменить набор if-else-if выражением switch:

```
// равноценно if-else-if выше
switch (num_tries)
{
case 1:
cout << "У-ух! Хорошая попытка, но не совсем\n"
break;
case 2:
cout << "М-да. Извини. Опять неправильно.\n";
break;

case 3:
cout << "А-а, это труднее, чем могло показаться, не так
ли?\n";
break;

default:
cout << "Похоже, дальше бесполезно!\n";
break;

}
```

Ключевое слово switch с последующим выражением в круглых скобках (да, имя объекта работает, как выражение). Выражение должно вычисляться как целое значение. Серия меток case, следующая за ключевым словом switch, определяет постоянное выражение. Результат выражения сравнивается с каждой из меток case по очереди. Если есть совпадение, выражение, следующее за case, выполняется. Если совпадений нет, и метка default присутствует, выполняется выражение, следующее за ней. Если же нет ни совпадений, ни метки default, не делается ничего.

Почему я поместил выражение break в конце каждой метки case? Каждая метка case проверяется по очереди на совпадение со значением выражения. Каждая несовпадающая метка case пропускается по очереди. Когда обнаруживается совпадение, выражение, следующее за меткой, выполняется. Увы, выполнение продолжается со следующими метками, пока не будет достигнут конец меток. Если значение num_tries,

например, равно 2, и если нет выражений break, вывод будет выглядеть так:

```
// output if num_tries == 2 and
// we had forgotten the break statements
М-да. Извини. Опять неправильно.
А-а, это труднее, чем могло показаться, не так ли?
Похоже, дальше бесполезно!
```

Вторая метка case совпадает, все метки case, следующие за совпадающей, также выполняются, пока не завершается оператор. Этот алгоритм дает выражение break. Вы можете спросить: почему выражение switch так устроено? Вот пример этого провала, который придется в самый раз:

```
switch (next_char)
{  case "a": case "A":
   case "e": case "E":
   case "i": case "I":
   case "o": case "O":
   case "u": case "U":
++vowel_cnt;
break;// ...}
```

Циклы

Циклы выполняют выражения или блоки выражений до тех пор, пока выражение условия не становится истинным. Наша программа требует двух циклов (один вложен в другой). Пока пользователь желает угадывать последовательности:

```
{ выводить на дисплей последовательности
пока догадка неверна, а пользователь желает угадать еще
раз}
```

цикл while в C++ вполне подходит для нашей цели:

```
bool next_seq = true; // показать следующую
последовательность?
bool go_for_it = true; // пользователь хочет продолжить
угадывание?
bool got_it = false; // пользователь угадал?
int num_tries = 0; // количество попыток
int num_right = 0; // количество правильных ответов
```

```
while (next_seq == true){
// вывести на дисплей последовательность для пользователя
   while ((got_it == false) &&(go_for_it == true)){
    int usr_guess;
    cin >> usr_guess;
    num_tries++;
    if (usr_guess == next_elem) {
     got_it = true;
     num_cor++;
     }
   else {
      // пользователь не угадал
      // сказать, что ответ неверен
      // спросить, еще угадываем?
      if (usr_rsp == "N" || usr_rsp == "n") go_for_it =
false;
      }
} // конец вложенного цикла while
 cout << "Want to try another sequence? (Y/N) "
 char try_again;
 cin >> try-again;
 if (try_again == "N" || try_again == "n")next_seq =
false;
} // конец while(next_seq == true)
```

Цикл while начинается с вычисления выражения условия в круглых скобках. Если оно истинно, выражение или блок выражений, следующий за циклом while, выполняется. После выполнения выражения (блока) выражение условия обновляется. Этот цикл обновления/выполнения продолжается, пока условие не станет ложным. Обычно некоторое условие внутри исполняемого блока устанавливает условное выражение в состояние false. Если выражение условия никогда не становится ложным, мы говорим, что ошибочно попали в бесконечный цикл.

Наш внешний цикл while выполняется, пока пользователь не скажет, что хочет остановиться:

```
bool next_seq = true;
while (next_seq == true){ // ...
 if (try_again == "N" || try_again == "n")next_seq =
false;
}
```

где при инициализации next_seq в false блок выражений не будет выполняться. Вложенный цикл while позволяет пользователю выполнять множество аналогичных попыток.

Цикл может быть прерван внутри выполняемого блока выполнением выражения break. В следующем фрагменте кода цикл while выполняется, пока tries_cnt не станет равно max_tries. Однако если пользователь дает правильный ответ, цикл прерывается выражением break:

```
int max_tries = 3;
int tries_cnt = 0;
while (tries_cnt < max_tries){ // прочитываем попытку
пользователя
 if (usr_guess == next_elem)
 break; // прерываем цикл
 tries_cnt++;// more stuff
}
```

Программа может иметь быстрое завершение текущего выполнения цикла, выполнив выражение continue. Например, рассмотрим следующий фрагмент программы, в котором все слова, в которых больше четырех букв, отбрасываются:

```
string word;
const int min_size = 4;
while (cin >> word){
if (word.size() < min_size)
   // прерываем этот цикл
   continue;
   // попадаем сюда, только если слово
   // больше или равно min-size ...
process_text(word);}
```

Если слово меньше min_size, выполняется выражение continue. Результатом выполнения выражения continue является прерывание текущего прохождения цикла. Остаток тела цикла while (в данном случае – process_text()) не выполняется. Вернее, цикл начинается заново, с новым вычислением условия, которое прочитывает другое значение строки в word. Если word больше или равно min_size, полное тело цикла выполняется. В этом случае все слова, имеющие больше четырех букв, отбрасываются.

Как использовать массивы и векторы

Ниже приведены первые восемь элементов из шести числовых последовательностей:

```
Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21
Lucas: 1, 3, 4, 7, 11, 18, 29, 47
Pell: 1, 2, 5, 12, 29, 70, 169, 408
Triangular: 1, 3, 6, 10, 15, 21, 28, 36
Square: 1, 4, 9, 16, 25, 36, 49, 64
Pentagonal: 1, 5, 12, 22, 35, 51, 70, 92
```

Наша программа должна выводить на дисплей пары элементов из последовательности и позволить пользователю угадать следующий элемент. Если пользователь угадывает и желает продолжить, программа должна вывести на дисплей следующую пару элементов, затем третью, и так далее. Как мы можем это сделать?

Если следующая пара берется из той же последовательности, пользователь, разгадавший одну пару, угадает их все. Это не интересно. Так что будем брать следующую пару из другой числовой последовательности при каждом проходе основного цикла программы.

Теперь будем выводить на дисплей максимум шесть пар элементов за сессию: по одной паре из каждой из шести последовательностей. Мы постараемся сделать это так, чтобы при выводе на дисплей пары элементов не знать, из какой числовой последовательности будет взята пара при следующем проходе цикла. Каждый проход должен иметь доступ к трем значениям: паре элементов и элементу, следующему за ней в последовательности.

Решение, которое мы обсудим в этом разделе, использует контейнерный тип, способный поддержать смежную последовательность целых значений, которые могут быть доступны не по имени, а по позиции в контейнере. Мы запомнили 18 значений в контейнере как подборку шести групп: первые два в группе представляют пару для вывода на дисплей, третий – следующий элемент в последовательности. При каждом проходе цикла мы добавляем три индексных значения, проходя по шести группам по очереди.

В C++ мы можем определить контейнер либо как встроенный массив, либо как вектор класса стандартной библиотеки. В основном, я рекомендую использовать класс векторов, а не встроенные массивы. Однако есть резон использовать массив, и важно понять, как использовать оба варианта.

Для определения встроенного массива мы должны обозначить тип элементов массива, дать ему имя и обозначить размер – количество элементов, которые массив должен содержать. Размер должен быть константой, то есть выражением, которое не меняется во время выполнения программы. Например, следующий код объявляет pell_seq массивом из 18 целых элементов.

```
const int seq_size = 18;
int pell_seq[seq_size];
```

Для определения объектов класса векторов мы должны вначале включить файл заголовка vector. Класс векторов – шаблон, поэтому мы определяем тип элементов в угловых скобках, следующих за именем класса. Размер помещается в круглых скобках, он не обязательно должен быть постоянным выражением. Следующий код определяет pell_seq как объект класса векторов, содержащий 18 элементов типа int. По определению каждый элемент инициализируется в «0».

```
#include <vector>

vector<int> pell_seq(seq_size);
```

Мы добираемся до элементов: массива или вектора, определяя его позицию в контейнере. Этот элемент индексируется с использованием оператора списка индексов ([]). Одна потенциальная «незадача» в том, что первый элемент находится в позиции «0», а не «1». Последний элемент индексируется на 1 меньше, чем размер контейнера. Для pell_seq правильные индексы – от 0 до 17, а не от 1 до 18. (Эта ошибка настолько распространена, что заслуживает иметь собственное имя: печально известная off-by-one-ошибка). Например, для получения первых двух элементов последовательности Pell мы пишем:

```
pell_seq[0] = 1; // присваиваем 1 первому элементу
pell_seq[1] = 2; // присваиваем 2 второму элементу
```

Вычислим следующие десять элементов последовательности Pell. Для прохождения через элементы вектора или массива мы обычно используем цикл for – другой базовый цикл C++. Например,

```
for (int ix = 2; ix < seq_size; ++ix)
pell_seq[ix] = pell_seq[ix - 2] + 2*pell_seq[ix - 1];
```

Цикл for состоит из следующих элементов:

```
for (начальное значение; условие; индексация) выражение;
```

Начальное значение выполняется единожды перед выполнением цикла. В нашем примере ix инициализируется как «2» перед началом выполнения цикла.

Условие служит для контроля над циклом. Оно вычисляется перед каждой итерацией цикла. Так что, сколько итераций при вычисленном условии равном true, столько раз выражение выполняется. Выражение может быть единственным или блочным. Если первое условие не удовлетворяется, выражение не выполнится никогда. В нашем примере условие проверяет, меньше ли ix, чем seq_size?

Индексация вычисляется после каждой итерации цикла. Она обычно используется для модификации начального значения объекта и проверяется в условии. Если первое вычисление условия принимает значение false, индексация не выполнится никогда. В нашем случае ix увеличивается с каждой итерацией цикла.

Для вывода элементов мы проходим через следующие операции:

```
cout << "The first " << seq_size<< " elements of the
Pell Series:\n\t";
for (int ix = 0; ix < seq_size; ++ix)cout <<
pell_seq[ix] << " ";
cout << "\n";
```

при желании мы можем обойтись без начального значения, индексации или (реже) условия для цикла for. Например, мы можем переписать предыдущий цикл как:

```
int ix = 0; // ...
for (; ix < seq_size; ++ix)// ...
```

Точка с запятой необходима, чтобы показать пустое начальное значение.

Наш контейнер содержит второй, третий и четвертый элементы каждой из шести последовательностей. Как мы заполним контейнер подходящими значениями? Встроенный массив может специфицироваться инициализационным списком, содержащим список значений, разделенных запятыми для всех элементов или подмножества элементов:

```
int elem_seq[seq_size] = {
1, 2, 3, // Fibonacci
3, 4, 7, // Lucas
2, 5, 12, // Pell
3, 6, 10, //Triangular
4, 9, 16, // Square
5, 12, 22 // Pentagonal
};
```

Количество значений, записанных в список инициализации, не должно превышать размера массива. Если мы предлагаем меньше значений, чем размер массива, недостающие элементы инициализируются как «0». При жедании мы можем позволить компилятору вычислить размер массива на основании количества значений, которые включаем в список:

```
// компилятор рассчитает размер в 18 элементов
int elem_seq[] = {1, 2, 3, 3, 4, 7, 2, 5, 12,3, 6, 10,
4, 9, 16, 5, 12, 22};
```

Класс векторов не поддерживает список инициализации. Несколько нудным решением будет присвоение значения каждому элементу отдельно:

```
vector<int> elem_seq(seq_size);
elem_seq[0] =1;
elem_seq[1] =2;
// ...
elem_seq[17] =22;
```

Еще одна альтернатива – инициализировать встроенный массив и использовать его для инициализации вектора:

```
int elem_vals[seq_size] = {1, 2, 3, 3, 4, 7, 2, 5, 12,3,
6, 10, 4, 9, 16, 5, 12, 22 };
```

```
// инициализация elem_seq значениями elem_vals
vector<int> elem_seq(elem_vals, elem_vals+seq_size);
```

elem_seq получает два значения. Эти значения в действительности адресованные – они отмечают диапазон элементов, с которым вектор будет инициализирован. В этом случае мы отметили 18 элементов, содержащихся в elem_vals и копируемых в elem_seq.

А теперь посмотрим, как мы можем использовать elem_seq. Одно различие между встроенными массивами и классом векторов состоит в том, что вектор знает свой размер. Наш предыдущий цикл for проходил через встроенный массив. Посмотрим, велика ли разница при использовании вектора:

```
// elem_seq.size() возвращает число элементов
// содержащихся в векторе elem_seq
cout << "The first " << elem_seq.size()<< " elements of
the Pell Series:\n\t";
for (int ix = 0; ix < elem_seq.size(); ++ix)
cout << pell_seq[ix] << " ";
```

cur_tuple представляет собой индекс в текущей последовательности, выводимой на дисплей. Мы инициализируем его в «0». С каждым проходом цикла мы добавляем «3» в cur_tuple, устанавливая его для индексации первого элемента следующей последовательности для вывода на дисплей:

```
int cur_tuple = 0;
while (next_seq == true &&cur_tuple < seq_size) {
cout << "The first two elements of the sequence are: "
<< elem_seq[cur_tuple] << ", "
<< elem_seq[cur_tuple + 1]
<< "\nWhat is the next element? ";
 // ...
 if (usr_guess == elem_seq[cur_tuple + 2])
 // правильно!

 // ...
 if (usr_rsp == "N" || usr_rsp == "n")next_seq = false;
 else cur_tuple += 3;
}
```

Полезно было бы сохранять путь к последовательности, которая в настоящий момент активна. Запомним имя каждой последовательности как строку:

```
const int max_seq = 6;
string seq_names[max_seq] = {"Fibonacci","Lucas","Pell",
"Triangular","Square","Pentagonal"};
```

Мы можем использовать seq_names следующим образом:

```
if (usr_guess == elem_seq[cur_tuple + 2]) {
++num_cor;
cout << "Very good. Yes, "
   << elem_seq[cur_tuple + 2]
   << " is the next element in the "
   << seq_names[cur_tuple / 3] << "sequence.\n";
}
```

Выражение cur_tuple/3 получает по очереди 0, 1, 2, 3, 4 и 5, индексируя в массиве строк элементы, которые идентифицируют активную последовательность.

Указатели дают больше гибкости

Наше решение по выводу на дисплей в предыдущей секции имеет два основных недостатка. Во-первых, оно ограничено выводом шести числовых последовательностей – если пользователь угадает все шесть, программа сразу завершится. Во-вторых, она всегда выводит те же самые шесть пар элементов в той же последовательности. Как же увеличить гибкость программы?

Одно из возможных решений – создать шесть векторов, по одному на каждую последовательность, рассчитанных на одинаковое количество элементов. На каждом проходе цикла мы выбираем пары элементов из разных векторов. При повторном использовании вектора мы возьмем пару элементов из другой части вектора. Это приблизит нас к устранению обоих отмеченных недостатков.

Как и в предыдущем решении, хотелось бы иметь «прозрачный» доступ к разным векторам. В предыдущем разделе мы достигали «прозрачности» за счет доступа по индексам, а не по имени. На каждом проходе цикла мы увеличивали значение индекса на 3. Сам код оставался неизменным.

В этом разделе мы добьемся «прозрачности» тем, что будем обращаться к вектору косвенно, через указатель, вместо обращения по имени. Указатель вносит определенный уровень косвенности в программу. Вместо того чтобы работать с объектом напрямую, мы будем работать с указателем, сохраняющим адрес объекта. Мы определим указатель, который будет адресоваться вектору чисел целого типа. При каждой итерации цикла мы будем модифицировать указатель, адресуясь к разным векторам. Фактический код для манипуляций с указателями не изменится.

Использование указателей дает два преимущества нашей программе: увеличивает гибкость программы и добавляет уровень гибкости, отсутствующий при прямой работе с объектом. Этот раздел убедит вас в правдивости обоих утверждений.

Мы уже знаем, как определять объект. Следующее выражение определяет `ival` как объект целого типа `int`, инициализированного значением `1024`:

```
int ival = 1024;
```

Указатель сохраняет адрес объекта данного типа. Для определения указателя мы добавляем к имени типа звездочку:

```
int *pi; // pi указатель на объект типа int
```

`pi` – указатель на объект типа `int`. Как мы должны инициализировать его для указания на `ival`? Определением имени объекта, примерно так:

```
ival; // определяет значение ival
```

Мы определяем ассоциированное значение, в нашем случае `1024`. Для получения адреса объекта, а не его значения, мы добавляем оператор адресации (`&`):

```
&ival; // определяет адрес ival
```

Для инициализации `pi` как адреса `ival` запишем следующее:

```
int *pi = &ival;
```

Для доступа к объекту, адресуемому указателем, мы должны разыменовать указатель, что даст содержимое объекта по

адресу, содержащемуся в указателе. Для этого добавим звездочку к указателю, следующим образом:

```
// разыменовываем pi для доступа к объекту им
адресуемому
if (*pi != 1024) // читаем
*pi = 1024; // пишем
```

Сложность инициализации с использованием указателя, как вы видите, исходит от запутывающего синтаксиса. Виною тому двойственная природа указателя. Мы можем манипулировать адресом, содержащимся в указателе, а можем манипулировать объектом, на который он указывает. Когда мы пишем:

```
pi; // определяет адрес сохраняемый в pi
```

то фактически управляем указателем объекта. А когда мы пишем:

```
*pi; // определяет значение объекта адресуемого pi
```

то управляем объектом, адресуемым pi.

Вторая сложность в понимании указателей – возможность отсутствия адресуемого объекта. Например, когда мы пишем *pi, это может быть, а может и не быть причиной краха программы при выполнении. Если pi адресуется к объекту, разыменовывание pi работает совершенно правильно. Если же pi не адресуется к объекту, попытка разыменовать pi влечет непредсказуемое поведение программы во время выполнения. Это означает, что когда мы используем указатель, то должны быть уверены, что он адресуется к объекту, прежде чем сделаем попытку разыменовать его.

Указатель, который не адресуется к объекту, имеет адресуемое значение «0» (иногда его называют нулевым указателем). Любой тип указателя может быть инициализирован, или определен со значением «0»:

```
// инициализация каждого указателя без адресации к
объекту
int *pi = 0;
double *pd = 0;
string *ps = 0;
```

Для защиты от разыменовывания нулевого указателя мы проверяем указатель, чтобы убедится, что его адресуемое значение не равно «0». Например,

```
if (pi && *pi != 1024)*pi = 1024;
```

Выражение:

```
if (pi && ...)
```

становится истинным, только если pi содержит иной адрес, чем «0». Если оно ложно, оператор И не выполняется во втором выражении. Для проверки мы обычно пользуемся оператором логического отрицания НЕ:

```
if (! pi) // истинно, если pi установлено в 0
```

А вот наши шесть объектов векторов последовательностей:

```
vector<int> fibonacci, lucas, pell, triangular, square,
pentagonal;
```

На что похож указатель вектора объектов целого типа? Что ж, в общем, указатель имеет такую форму:

```
(тип_объекта_указывающего_на * имя_указателя_объекта)
type_of_object_pointed_to * name_of_pointer_object
```

Наш указатель адресует тип vector<int>. Назовем его pv и инициализируем нулем:

```
vector<int> *pv = 0;
```

pv может адресоваться каждому вектору последовательности по очереди. Конечно, мы можем определить pv адресацией к каждой последовательности:

```
pv = &fibonacci;// ...
pv = &lucas;
```

но этим приносится в жертву «прозрачность» кода. Альтернативное решение – запомнить адреса каждой последовательности в векторе. Этот прием позволяет нам добраться до них «прозрачно», через индекс:

```
const int seq_cnt = 6;
// массив seq_cnt указателей на
// объекты типа vector<int>
```

```
vector<int> *seq_addrs[seq_cnt] = {&fibonacci, &lucas,
&pell, &triangular, &square, &pentagonal};
```

seq_addrs – это встроенный массив элементов типа vector<int>*. seq_addrs[0] содержит адрес fibonacci вектора, seq_addrs[1] – адрес lucas вектора и т.д. Мы используем это для доступа к различным векторам через индекс, а не по имени:

```
vector<int> *current_vec = 0;
// ...
for (int ix = 0; ix < seq_cnt; ++ix) {
current_vec = seq_addrs[ix];
// вывод на дисплей всех элементов осуществляется
// косвенно через current_vec
}
```

Оставшейся проблемой с задуманной реализацией является полная предсказуемость. Последовательности всегда Fibonacci, Lucas, Pell... Мы бы хотели сделать вывод на дисплей последовательностей случайным. Это возможно с использованием стандартной библиотеки языка С функциями rand() или srand():

```
#include <cstdlib>

srand(seq_cnt);
seq_index = rand() % seq_cnt;
current_vec = seq_addrs[seq_index];
```

rand() и srand() – функции стандартной библиотеки, которые поддерживают псевдослучайную генерацию. srand() активизирует генератор с его параметрами. Каждый вызов rand() возвращает целое значение в диапазоне от 0 до максимального целого значения, представленного int. Мы должны это ограничить числами от 0 до 5, чтобы они были правильными индексами seq_addrs. Оператор остатка (%) гарантирует нам индексацию между 0 и 5. Файл заголовка cstdlib содержит объявление обеих функций.

Мы сохраняем указатель на класс объекта несколько иначе, чем делали это с указателем на объект встроенного типа, потому что класс объекта имеет связанное с ним множество

операций, которые мы можем вызвать. Например, для проверки, является ли первый элемент вектора fibonacci единицей, можно написать:

```
if (! fibonacci.empty() &&  (fibonacci[1] == 1))
```

Как мы могли бы осуществить ту же проверку через pv? Объединение fibonacci и empty() через точку называется *оператором выбора члена.* Оно используется для выбора операций класса через объект класса. Для выбора операции класса через указатель используем оператор-стрелку (->) выбора члена:

```
! pv->empty()
```

Поскольку указатель может адресоваться к отсутствующему объекту, перед тем как мы используем empty() через pv, необходимо проверить, что адресация не нулевая:

```
pv && ! pv->empty()
```

Окончательно для вызова оператора индексов мы должны разыменовать pv (понадобятся дополнительные скобки вокруг разыменованного pv из-за более высокого приоритета индексного оператора):

```
if (pv && ! pv->empty() && ((*pv)[1] == 1))
```

Запись и чтение файлов

Если пользователю случится запустить нашу программу повторно, было бы здорово, если бы счет сохранялся для обеих сессий. Чтобы это стало возможным, мы должны:

- записать имя пользователя и данные сессии в файл в конце сессии;
- прочитать данные предыдущей сессии в программу при ее повторном запуске.

Посмотрим, как мы можем это сделать.

Для чтения и записи в файл мы должны включить файл заголовка fstream:

```
#include <fstream>
```

Чтобы открыть файл для вывода, определим объект класса ofstream (an output file stream – поток вывода файла), передавая его имя в открываемый файл:

```
// seq_data.txt открыт на вывод
ofstream outfile("seq_data.txt");
```

Что происходит, когда мы объявляем outfile? Если его не существует, он создается и открывается на вывод. Если же он существует, то открывается на вывод, а все данные, которые в нем содержатся, игнорируются.

Если мы хотим добавить, а не замещать данные в существующем файле, необходимо открыть файл в режиме добавления.

Мы делаем это, передавая второе значение ios_base::app объекту ofstream:

```
// seq_data.txt открывается в append mode (режим
добавления)
// новые данные добавляются в конец файла
ofstream outfile("seq_data.txt", ios_base::app);
```

Файл может не открыться. Прежде чем записывать в него, нужно убедиться, что он открылся успешно. Простейший путь проверки – убедиться в истинности объекта класса:

```
// если outfile определяется, как  false,
// файл не может быть открыт
if (! outfile)
```

Если файл не может быть открыт, объект класса ofstream становится ложным. В этом примере мы предупредим пользователя, выводом сообщения cerr. cerr представляет стандартную ошибку. cerr, подобно cout, выводится на терминал пользователя. Разница в том, что вывод cerr не буферизуется, оно выводится сразу на терминал:

```
if (! outfile)
// по какой-то причине не открывается ...
cerr << "Бах! Не могу сохранить данные сессии!\n";
else
// ok: outfile открыт, давайте писать данные
outfile << usr_name << " "
   << num_tries << " "
   << num_right << endl;
```

Если файл открывается успешно, мы непосредственно выводим в него данные, как делаем это для объектов класса ostream cout и cerr. В этом примере мы пишем три значения в outfile, последние два отделены пробелами. Endl – предопределенный манипулятор, поставляемый библиотекой iostream.

Манипулятор выполняет несколько операций с iostream, отличных от записи и чтения данных. endl вставляет символ перевода на новую строку, а затем сбрасывает на диск выходной буфер. Другие предопределенные манипуляторы включают hex, отображающий на дисплее целое число в шестнадцатеричном виде, oct, который отображает целое в восьмеричном виде, и setprecision(n), устанавливающий точность отображения чисел с плавающей точкой в n.

Чтобы открыть файл на ввод, мы определяем объект класса ifstream (an input file stream – поток ввода в файл), передавая ему имя файла. Если файл не может быть открыт, объект класса ifstream определяется как ложный. Иначе, файл позиционируется в начало данных, записанных в него:

```
// infile открыт в output mode
ifstream infile("seq_data.txt");
int num_tries = 0;
int num_cor = 0;

if (! infile){
// по какой-то причине файл не открывается ...
// мы будем предполагать, что это новый пользователь
...}
else {
// ok: читаем каждую линию входного файла
// смотрим, играл ли пользователь раньше ...
// формат каждой линиии:
// name num_tries num_correct
// nt: количество попыток
// nc: количество отгадываний
string name;
int nt;
int nc;
while (infile >> name){
infile >> nt >> nc;
if (name == usr_name) {
```

```
// нашли!
cout << "С возвращением, " << usr_name<< "\nВаш текущий
счет " << nc  << " out of " << nt << "\nУдачи!\n";
num_tries = nt; num_cor = nc;}}}
```

Каждый проход цикла while прочитывает новую линию файла, пока не будет достигнут конец файла. Когда мы пишем:

```
infile >> name
```

возвращаемое значение входного выражения – объект класса, из которого мы читаем– infile в данном случае. Когда конец файла достигнут, условие true объекта класса сменяется на false. Это причина, по которой условное выражение цикла while прерывается, когда достигается конец файла:

```
while (infile >> name)
```

Каждая линия файла содержит строку, за которой следуют два целых в форме:

```
anna 24 19
danny 16 12 ...
```

Выражение:

```
infile >> nt >> nc;
```

читает по очереди количество попыток пользователя в nt и количество угадываний в nc.

Если мы хотим и читать, и писать в тот же самый файл, мы определяем объект класса fstream. Для открывания его в режиме дополнения мы должны передать второе значение в форме:

```
 ios_base::in|ios_base::app:
fstream iofile("seq_data.txt",ios_base::in|ios_base::app);
if (! iofile)
// файл не открывается по какой-то причине ... гад!
{ // переходим к началу файла для начала чтения
iofile.seekg(0);
// ok: все остальное без изменений ...}
```

Когда мы открываем файл в режиме добавления, текущая позиция – конец файла. Если мы пытаемся читать файл без перепозиционирования, то просто получаем конец файла. Оператор `seekg()` возвращает `iofile` к началу файла. Поскольку он открыт в режиме дополнения, любая операция записи добавляет данные в конец файла.

Ссылки на полезные сайты в Интернете

Микроконтроллеры PIC:
http://www.microchip.ru
Электронные компоненты и наборы, рассылка по почте наложенным платежом:
http://www.dessy.ru
http://www.chipinfo.ru
http://www.interlavka.narod.ru
Магазин «Чип и Дип»:
http://www.chip-dip.ru
Мастер Кит:
http://www.masterkit.ru
Системы «Умный дом»:
Система X-10 – http://www.ydom.ru
Система AMX – http://www.arhelect.ru; http://www.amxcorp.com
Система Instabus – http://www.gira.ru
Система Crestron – http://www.crestron.com
Программы для работы с контроллером:
MPLAB – http://www.microchip.ru
PonyProg2000 – http://www.LancOS.com
Компилятор «С» для MPLAB:
http://www.htsoft.com
Программа для работы с COM-портом:
http://www.software.rs232.ru
Программа WinLIRC:
http://winlirc.sourceforge.net

Сайты с полезными программами, схемами и идеями:

http://alex-uc.narod.ru
http://www.homeautomationindex.com
http://linuxha.sourceforge.net
http://www.ukrocketman.com
http://www.razumdom.ru
http://neolive.org/linux/application/
http://irls.narod.ru
http://www.qrz.ru/schemes/
http://rf.atnn.ru
http://kazus.ru/guide/index.html
http://www.cxem.net
http://vksn.narod.ru/links.html

